SPECIAL SYSTEMS & SOFTWARE TECHNOLOGY CONFERENCE ISSUE

CrossTalk

May 2006 The Journal of Defense Software Engineering Vol. 19 No. 5

Transforming: Business, Security, Warfighting

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
MAY 2006		**00-00-2006 to 00-00-2006**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
CrossTalk: The Journal of Defense Software Engineering. Volume 19, Number 5, May 2006	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **OO-ALC/MASE,6022 Fir Ave,Hill AFB,UT,84056-5820**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**Same as Report (SAR)**	**32**	

Transforming: Business, Security, Warfighting

4 **21st Century Processes for Acquiring 21st Century Software-Intensive Systems of Systems**

1 May, Track 2

This article summarizes characteristics of 21st century software-intensive systems of systems and indicates some of the major problems associated with using the traditional acquisition processes on them, while introducing new acquisition development processes to address 21st century software-intensive systems of systems.

by Dr. Barry Boehm and Jo Ann Lane

10 **Tackling the Cost Challenges of System of Systems**

1 May, Track 1

This article provides guidelines to decision makers performing high-level analysis of system of systems costs in order to pursue the most affordable solution to meet the mission needs.

by Arlene F. Minkiewicz

15 **Building Multilevel Secure Web Services-Based Components for the Global Information Grid**

4 May, Track 3

This article discusses using multiple independent levels of security for the Department of Defense's Global Information Grid and provides examples where it has been successfully used.

by Dr. Dylan McNamee, CDR Scott Heller, and Dave Huff

20 **Practical Performance-Based Earned Value**

2 May, Track 5

This update on Performance-Based Earned Value (PBEV) provides guidance on practical ways to implement two of the four PBEV principles.

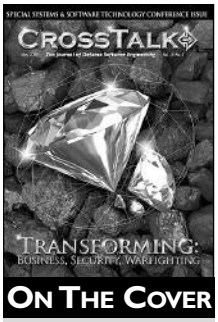
by Paul J. Solomon

25 **Lessons Learned Using Agile Methods on Large Defense Contracts**

1 May, Track 2

What is working, and what isn't working, when applying agile software development on large U.S. government defense projects? This article answers these questions by employing scenarios based on actual project situations that occurred in 2005 and shares the latest lessons learned from these scenarios.

by Paul E. McMahon



On The Cover

Cover Design by Kent Bingham.

Additional art services provided by Janna Jensen, M.P.C. jensendesigns@aol.com

Departments

3 **From the Sponsor**
From the Publisher

9 **Coming Events**

19 **Web Sites**

24 **Call for Articles**

30 **Visit CrossTalk at the SSTC**

31 **BackTalk**

CrossTalk

76 SMXG Co-Sponsor	*Kevin Stamey*
309 SMXG Co-Sponsor	*Randy Hill*
402 SMXG Co-Sponsor	*Bob Zwitch*
DHS Co-Sponsor	*Joe Jarzombek*
NAVAIR Co-Sponsor	*Jeff Scwalb*
Publisher	*Brent Baxter*
Associate Publisher	*Elizabeth Starrett*
Managing Editor	*Kase Johnstun*
Associate Editor	*Chelene Fortier-Lozancich*
Article Coordinator	*Nicole Kentta*
Phone	(801) 775-5555
E-mail	crosstalk.staff@hill.af.mil
CrossTalk Online	www.stsc.hill.af.mil/crosstalk

CrossTalk, The Journal of Defense Software Engineering is co-sponsored by the U.S. Air Force (USAF), the U.S. Department of Homeland Security (DHS), and the U.S. Navy (USN). USAF co-sponsors: Oklahoma City-Air Logistics Center (ALC) 76 Software Maintenance Group (SMXG), Ogden-ALC 309 SMXG, and Warner Robins-ALC 402 SMXG. DHS co-sponsor: National Cyber Security Division of the Office of Infrastructure Protection. USN co-sponsor: Naval Air Systems Command.

The USAF Software Technology Support Center (STSC) is the publisher of CrossTalk, providing both editorial oversight and technical review of the journal. CrossTalk's mission is to encourage the engineering development of software to improve the reliability, sustainability, and responsiveness of our warfighting capability.

S T S C

Subscriptions: Send correspondence concerning subscriptions and changes of address to the following address. You may e-mail us or use the form on p. 29.

309 SMXG/MXDB
6022 Fir AVE
BLDG 1238
Hill AFB, UT 84056-5820

Article Submissions: We welcome articles of interest to the defense software community. Articles must be approved by the CrossTalk editorial board prior to publication. Please follow the Author Guidelines, available at <www.stsc.hill.af.mil/crosstalk/xtlkguid.pdf>. CrossTalk does not pay for submissions. Articles published in CrossTalk remain the property of the authors and may be submitted to other publications.

Reprints: Permission to reprint or post articles must be requested from the author or the copyright holder and coordinated with CrossTalk.

Trademarks and Endorsements: This Department of Defense (DoD) journal is an authorized publication for members of the DoD. Contents of CrossTalk are not necessarily the official views of, or endorsed by, the U.S. government, the DoD, or the STSC. All product names referenced in this issue are trademarks of their companies.

Coming Events: Please submit conferences, seminars, symposiums, etc. that are of interest to our readers at least 90 days before registration. Mail or e-mail announcements to us.

CrossTalk Online Services: See <www.stsc.hill.af.mil/crosstalk>, call (801) 777-0857 or e-mail <stsc.webmaster@hill.af.mil>.

Back Issues Available: Please phone or e-mail us to see if back issues are available free of charge.

From the Sponsor

Transformation: A Continuous Process

The 2006 Systems and Software Technology Conference and this month's issue of CrossTalk have as their theme "Transforming: Business, Security, Warfighting." Transformation is not just the current buzzword: Many industries and the military have realized that through the investment of transformation efforts, they can strategically posture themselves for the future.

A.K. Cebrowski, the Director of the Office of Transformation for the Office of the Secretary of Defense, stated, "Over the long term, our security and prospects for peace and stability for much of the rest of the world depend on the success of our transformation." Today, our military faces an ever-growing number of emerging security threats. The conflicts are varied – spread out over the globe, continuous, and longer lasting. We are moving from threat-based to capabilities-based planning. The focus is on effects required – not the number of targets destroyed.

Benjamin Franklin stated, "When you're finished changing, you're finished." Clearly, a poignant thought that we must always continue to change, develop, and improve. Transformation is a continuous process, not an end point. It has conceptual, cultural, organizational, and technological dimensions. Being transformational implies that we must continually adapt to a changing environment and that we be innovative, adaptive, and responsive. We have to be easy to do business with. We have to be effective and efficient.

Transformation requires that leaders be prepared for change, and that we invest in new technologies. Leaders have to encourage new ways of thinking; sometimes this includes using old capabilities in new ways. Gen. Richard B. Myers, former chairman of the Joint Chiefs of Staff, said, "In today's world, there ought to be a premium for people who are thinking, innovative, and are willing to take appropriate risks. If you don't try, and you stay locked in the doctrine that brought you there, you're going to fail."

The featured articles in this issue of CrossTalk develop many valuable concepts to transform our business practices. These articles certainly offer concepts for more agile business practices, and better cost and performance results: Great information to help us all strategically posture ourselves for the future.

Bob Zwitch
Warner Robins Air Logistics Center Co-Sponsor

From the Publisher

Education Is Key for Successful Transformation

As Mr. Zwitch discusses, the Department of Defense (DoD) continues to transform to ensure our enduring success. In addition to finding improved ways of doing things, transformation requires educating the members of the DoD about those improvements. CrossTalk's parent organization, the Software Technology Support Center, was established for just this purpose. As part of our endeavor to educate software practitioners on methods to better acquire and develop software, the Systems and Software Technology Conference (SSTC) and CrossTalk were established. Once a year, SSTC and CrossTalk join forces to share improvements in person and in print. All the articles in this month's issue will be discussed in presentations at SSTC 2006.

The articles in this month's issue include an update to the spiral model that addresses acquiring systems of systems; a discussion on the special cost challenges that systems of systems create; one approach for better securing the Global Information Grid; examples for using Performance-Based Earned Value; and tips to help large projects better use agile software development methods. I hope you enjoy this month's issue of CrossTalk, and hope you enjoy the presentations at SSTC.

Elizabeth Starrett
Associate Publisher

21st Century Processes for Acquiring 21st Century Software-Intensive Systems of Systems

Dr. Barry Boehm and Jo Ann Lane
University of Southern California

Monday, 1 May 2006
Track 2: 8:00 – 11:15 a.m.
Ballroom B

Our experiences in helping to define, acquire, develop, and assess 21st century software-intensive systems of systems (SISOS) have taught us that traditional 20th century acquisition and development processes do not work well on such systems. This article summarizes the characteristics of such systems, and indicates the major problem areas in using traditional processes on them. We also present new processes that we and others have been developing, applying, and evolving to address 21st century SISOS. These include extensions to the risk-driven spiral model to cover broad (many systems), deep (many supplier levels), and long (many increments) acquisitions needing rapid fielding, high assurance, adaptability to high-change traffic, and complex interactions with evolving commercial off-the-shelf products, legacy systems, and external systems.

Between now and 2025, the ability of organizations and their products, systems, and services to compete, adapt, and survive will depend increasingly on software and the ability to integrate related software-intensive systems into systems of systems (SOS). As is being seen in current products (automobiles, aircraft, radios) and services (financial, communication, defense), software provides both competitive differentiation and rapid adaptability to competitive change. It facilitates rapid tailoring of products and services to different market sectors and rapid and flexible supply chain management.

The resulting software-intensive systems and SOS face ever-increasing demands to provide safe, secure, and reliable systems; provide competitive discriminators in the marketplace; support the coordination of multi-cultural global enterprises; enable rapid adaptation to change; and help people cope with complex masses of data and information. These demands will cause major differences in the processes currently used to define, design, develop, deploy, and evolve a diverse variety of software-intensive systems and software-intensive SOS (SISOS).

SISOS Trends and Their Influence on Systems and Software Engineering Processes

Today's trend towards larger, software-intensive systems and SOS often require much more complex systems and software engineering processes and better integration of these processes across the systems engineering and software engineering organizations. This section provides an overview of key SISOS historical trends, features, development organizations, and potential pitfalls.

Historical Evolution of Processes

Historically (and even recently for some forms of agile methods), systems and software development processes and maturity models were recipes for standalone *stovepipe* systems with high risks of inadequate interoperability with other stovepipe systems. Experience has shown that such collections of stovepipe systems cause unacceptable delays in service, uncoordinated and conflicting plans, ineffective or dangerous decisions, and an inability to cope with rapid change.

During the 1990s and early 2000s, standards such as the International Organization for Standardization (ISO)/International Electrotechnical Commission (IEC) 12207 [1] and ISO/IEC 15288 [2] began to emerge that situated systems and software project processes within an enterprise framework. Concurrently, enterprise architectures such as IBM Zachman Framework [3], Reference Model for Open Distributed Processing (RM-ODP), [4] and the U.S. Federal Enterprise Architecture Framework [5], have been developing and evolving along with a number of commercial Enterprise Resource Planning (ERP) packages.

These frameworks and support packages are making it possible for organizations to reinvent themselves around transformational, network-centric SOS. As discussed in [6], these are necessary SISOS that have equally tremendous opportunities for success and risks of failure. Examples of successes are Federal Express; Wal-Mart; and the U.S. Command, Control, Intelligence, Surveillance, and Reconnaissance (C2ISR) system in Iraq. Examples of failures are the Confirm travel reservation system; K-Mart; and the U.S. Advanced Automation System for air traffic control. ERP packages have been the source of many successes and many failures, implying the need for considerable risk/opportunity assessment before committing to an ERP-based solution.

Table 1: *Software-Intensive Systems of Systems (SISOS) Solution Spaces*

Characteristic	Range of Values
Size	10-100 million lines of code
Number of external interfaces	30-300
Number of *coopetitive* suppliers	20-200
Depth of supplier hierarchy	6-12 levels
Number of coordination groups	20-200

Key SISOS Features

There are many definitions of SOS [7]. For this article, the distinguishing features of SOS are not only that they integrate multiple, independently developed systems, but also that they are very large, dynamically evolving, and unprecedented with emergent requirements and behaviors, and complex socio-technical issues to address.

Table 1 provides some additional characteristics of SISOS.

SISOS Development Organization Trends and Issues

There is often a lead system integrator who is responsible for developing SOS architecture, identifying the suppliers and vendors to provide various SOS components, adapting the architecture to meet evolving requirements and selected vendor limitations or constraints, overseeing the implementation efforts, and planning and executing the SOS level integration and test activities.

Keys to successful SOS development are the ability to: achieve timely decisions with a potentially diverse set of stakeholders; quickly resolve conflicting needs; and coordinate the activities of multiple vendors who are currently working together to provide capabilities for the SOS, but are often competitors on other system development efforts (sometimes referred to as "coopetitive" relationships).

Potential SISOS Pitfalls

Our work in supporting SISOS development programs has shown that using a risk-driven spiral process with early attention to SISOS risks and systems architecting methods can avoid many of the SISOS development pitfalls [8]. A prioritized list of the top 10 SISOS risks we have encountered includes the following:

1. Acquisition management and staffing.
2. Requirements/architecture feasibility.
3. Achievable software schedules.
4. Supplier integration.
5. Adaptation to rapid change.
6. Systems and software quality factor achievability.
7. Product integration and electronic upgrade.
8. Commercial off-the-shelf (COTS) software and reuse feasibility.
9. External interoperability.
10. Technology readiness.

Strategies for addressing these risks are described in [8].

A Scalable Spiral Process Model for 21st Century SISOS

In applying risk management to the set of risks described above, the outlines of a hybrid plan-driven/agile process for developing SISOS product architecture are emerging. To keep SISOS developments from becoming destabilized from large amounts of change traffic, it is important to organize development into plan-driven increments in which the suppliers develop to interface specifications that are kept stable by deferring changes, so that the systems can plug and play at the end of the increment (nobody has yet figured out how to do daily builds for these kinds of systems).

However, for the next increment to hit the ground running, an extremely agile team needs to be concurrently doing a continuous market, competition, and technology watch; change impact analysis; COTS refresh; and renegotiation of the next increment's prioritized content and the interfaces between the suppliers' next-increment interface specifications. This requires new approaches not only to process management, but also to staffing and contracting. The following sections elaborate on this emerging process architecture and its challenges.

21st Century SISOS Development and Evolution Modes

In the next 10 to 20 years, several 21st century system and software development and evolution modes will have emerged as the most cost-effective ways to develop needed capabilities in the context of the trends discussed earlier. The four most common modes are likely to be exploratory development of unprecedented capabilities, business model-based user programming, hardware and software product lines, and network-centric SOS that will necessarily be software-intensive [6]. There are new challenges for organizations in the process of transforming themselves from collections of weakly coordinated, vertically integrated stovepipe systems into seamlessly interoperable network-centric SOS (NCSOS).

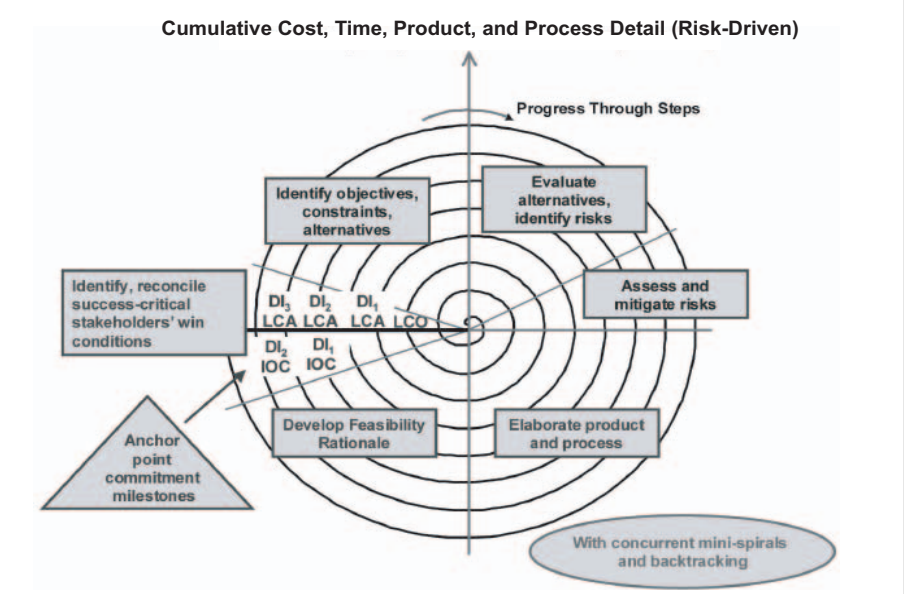
Architectures of these NCSOS are highly software-intensive and need to be simultaneously robust, scalable, and evolvable in flexible but controllable ways. The NCSOS development projects need processes such as the Internet spiral development process [9], but due to competitive pressures, their

Spiral 2005 Process Model

The spiral model, first introduced in 1986, has continued to evolve to meet the needs of our evolving development processes. The figure below shows the latest version of the spiral model. Keys to this model are the use of success-critical stakeholder win-win negotiations to determine the system's objectives, constraints, and primary solution alternatives; and the life cycle anchor points that are the common system/software stakeholder commitment points for concurrent engineering. (These often replace the more traditional sequential Department of Defense system development milestone reviews such as System Requirements Review, Preliminary Design Review, and Initial Operational Capability.) The life cycle anchor points include:

- **Life Cycle Objectives (LCO):** The stakeholders' commitment to support system architecting.
- **Life Cycle Architecture/Development Increments ($\frac{DI_N}{LCA}$):** The stakeholders' commitment to support the full life-cycle development.
- **Incremental Operational Capabilities/Development Increments ($\frac{DI_N}{IOC}$):** The stakeholders' commitment to support operations.

The 2005 spiral framework provides a common frame of reference for the process views that we will subsequently describe.



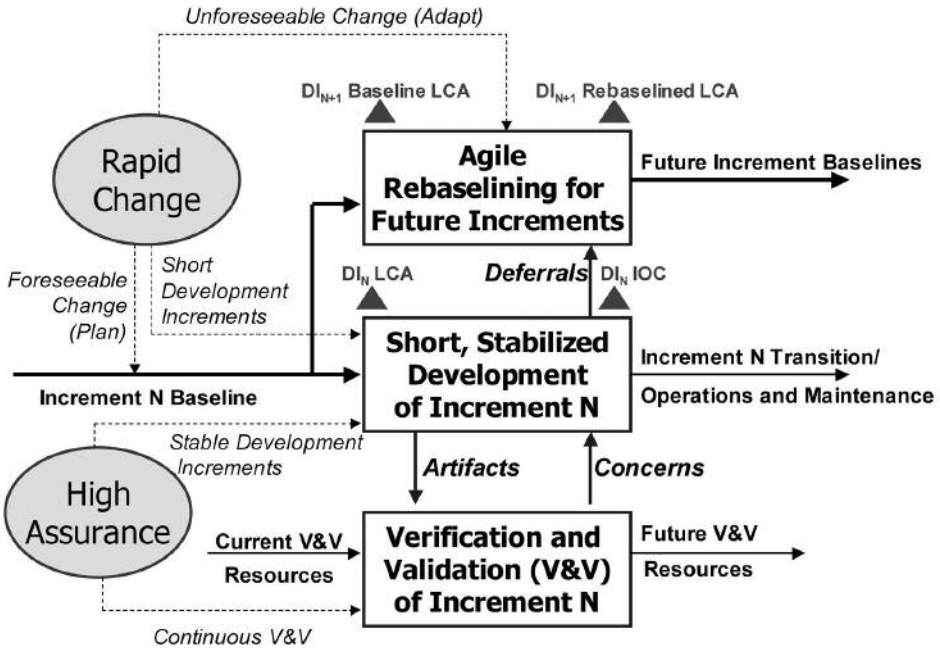


Figure 1: *The Scalable Spiral Process Model – Increment Activities*

processes must generally operate on much tighter timescales than were involved in the early evolution of the Internet.

Evolutionary development, business model-based user programming, and hardware and software product line development as described in [10] are key trends for the development and evolution of SISOS components. In the following sections, we describe an emerging, scalable spiral process model for developing and evolving 21st century product lines and NCSOS.

Overview of the Scalable Spiral Process Model

Complex systems and complex processes require multiple views to understand various aspects of the system and its development processes. And these multiple views require some type of anchor points to help relate one view to another. Our first view begins with the spiral model view shown in the sidebar (see page 5).

Based on our experiences in adapting the spiral model to the development of SISOS representative of the 21st century trends discussed earlier, we have been converging on a scalable spiral process model. This model has shown in partial implementation to date to scale well from small e-services applications to super-large defense SOS and multi-enterprise supply chain management systems. The model contains familiar elements, but organizes them in ways that involve new approaches to enterprise organization, contracting, incentives, staffing, education, and career development. Figure 1 shows a single increment of the development and evolution portion of the model. It assumes that the organization has developed the following:

- A best-effort definition of the system's steady-state capability.
- An incremental sequence of prioritized capabilities culminating in the steady-state capability.
- A feasibility rationale providing sufficient feasibility evidence for each increment and the overall system. This evidence should show that system architecture will support the incremental capabilities, that each increment can be developed within its available budget and schedule, and that the series of increments create a satisfactory return on investment for the organization and mutually satisfactory outcomes for the success-critical stakeholders.

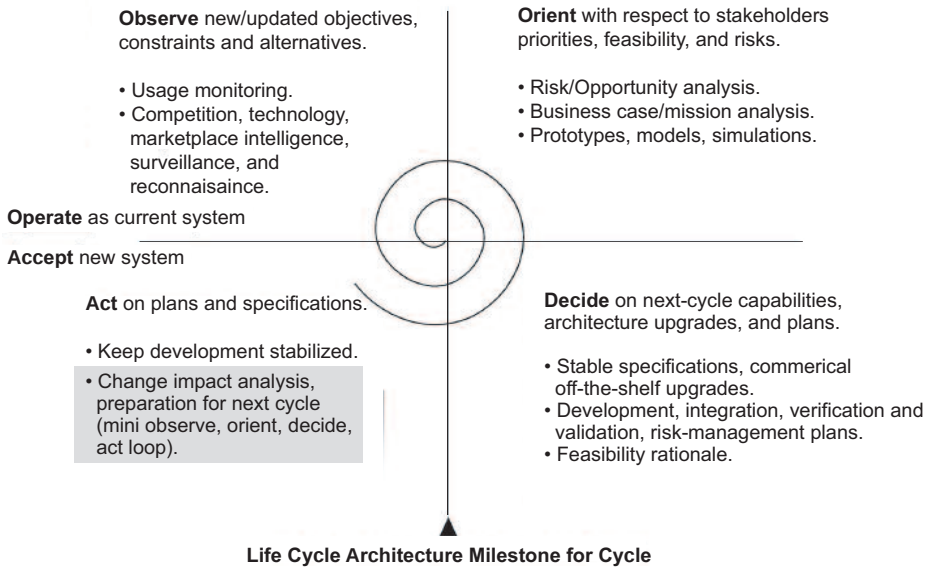
As seen in Figure 1, the model is organized to simultaneously address conflicting 21st century challenges of rapid change and high assurance of dependability. It also addresses the need for rapid fielding of incremental capabilities with a minimum of rework, and the other major 21st century trends involving integration of systems and software engineering, COTS components, legacy systems, globalization, and user value considerations [10].

The need to deliver high-assurance incremental capabilities on short, fixed schedules means that each increment needs to be kept as stable as possible. This is particularly the case for a large SOS with deep supplier hierarchies (often six to 12 levels) in which a high level of rebaselining traffic can easily lead to chaos. In keeping with the use of the spiral model as a risk-driven process model generator, the risks of destabilizing the development process make this portion of the project into a waterfall-like, build-to-specification subset of the spiral model activities. The need for high assurance of each increment also makes it cost-effective to invest in a team of appropriately skilled personnel to continuously verify and validate the increment as it is being developed.

However, the previous discussion on *deferring change* does not imply deferring change impact analysis, change negotiation, and rebaselining until the beginning of the next increment. With a single development team and rapid rates of change, this would require a team optimized to develop stable plans and specifications to spend much of the next increment's scarce calendar time performing tasks better suited to agile teams. Instead, Figure 1 shows how the spiral project would organize itself as follows:

- A plan-driven team transforms a build-

Figure 2: *Observe, Orient, Decide, Act (OODA) Loop*



to DI_1 life-cycle architecture (LCA) package of validated specifications and plans (using one or more spiral cycles or intermediate builds) into a completed initial operational capability deliverable.

- Meanwhile, an independent verification and validation (IV&V) team continually verifies and validates the plan-driven increment under development.
- Meanwhile, an agile team adjusts and rebaselines the build-to specifications and plans for the next increment (DI_2) for hand-off to the plan-driven team.

The process in Figure 1 is then applied similarly to the subsequent cycles in the spiral chart.

The appropriate metaphor for addressing rapid change is not a build-to-specification metaphor or a purchasing-agent metaphor, but is an adaptive *C2ISR* metaphor as shown in Figure 2. It involves an agile team performing the first three activities of the C2ISR *Observe*, *Orient*, *Decide*, *Act* (OODA) loop for the next increments, while the plan-driven development team is performing the *Act* activity for the current increment. *Observing* involves monitoring changes in relevant technology and COTS products in the competitive marketplace, in external interoperating systems, and in the environment; and monitoring progress on the current increment to identify slowdowns and likely scope deferrals. *Orienting* involves performing change impact analysis, risk analysis, and trade-off analysis to assess candidate rebaselining options for upcoming increments. *Deciding* involves stakeholder renegotiation of the content of upcoming increments, architecture rebaselining, and the degree of COTS upgrading to be done to prepare for the next increment. It also involves updating the future increments' feasibility rationales to ensure that renegotiated scopes and solutions can be achieved within budget and schedule. The LCA milestone at the bottom of Figure 2 corresponds with the DI_{N+1} Rebaselined LCA increment in Figure 1.

A successful rebaseline means that the plan-driven development team can hit the ground running at the beginning of the Act phase of developing the next increment, and the agile team can hit the ground running on rebaselining definitions of the increments beyond. Figure 3 shows how this three-team cycle (lean, plan-driven, stabilized developers; thorough IV&V people; and agile, proactive rebaseline people) plays out from one increment to the next, including the early product line or SOS inception and elaboration phases with their pass-fail, life-cycle objectives and LCA exit milestones. The

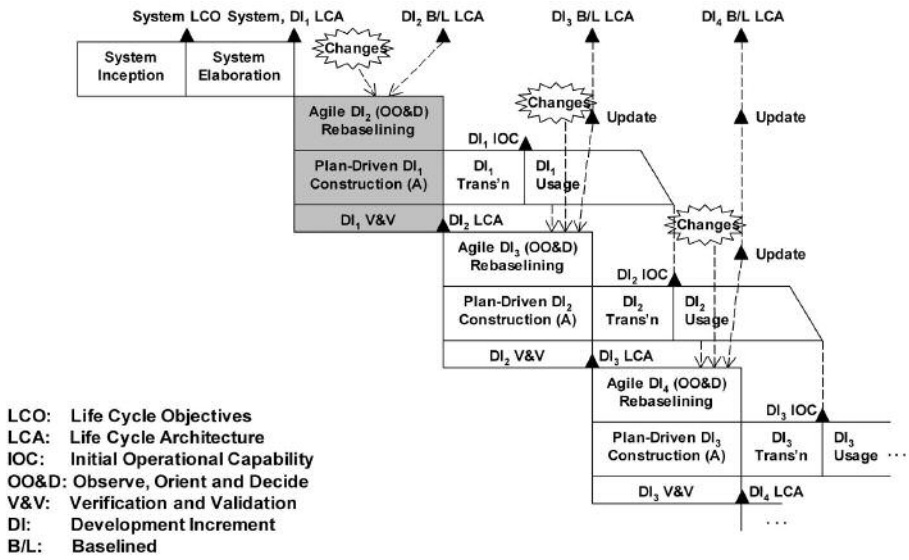


Figure 3: *The Scalable Spiral Process Model: Life Cycle View*

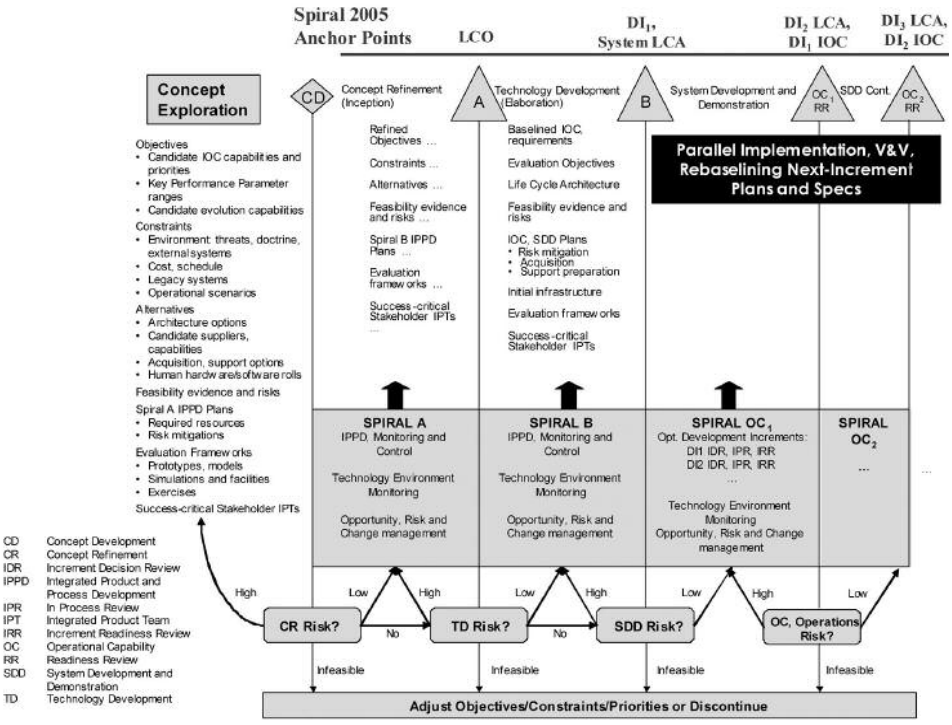
shaded activities in Figure 3 are the same set of activities that are shown in detail in Figure 1. Note that *OO&D* in each agile rebaselining increment stands for observe, orient, and decide, and *not* object-oriented design. The (A) below it stands for the *Act* portion of the OODA loop for the current increment. Note also that, as much as possible, usage feedback from the previous increment is not allowed to destabilize the current increment, but is fed into the definition of the following increment. Of course, some level of mission-critical updates will need to be fed into the current increment, but only when the risk of not doing so is greater than the risk of destabilizing the current increment.

As with command and control, the OO&D rebaselining portion of the project is not a sequential waterfall process. Instead, it is a risk-driven set of concurrent prototyping, analysis, and stakeholder renegotiation activities that lead to a best-possible redefinition of plans and specifications to be used by the stabilized development team for the next increment. For people familiar with the Department of Defense 5000 series of acquisition milestones, Figure 4 provides a mapping of them onto the Spiral 2005 anchor points.

Acquisition as C2ISR Versus Purchasing

The 20th century purchasing agent or contracts manager is most comfortable with a

Figure 4: *Spiral 2005 Anchor Points in Relation to Department of Defense 5000 Milestones*



fixed procurement to a set of pre-specified requirements; selection of the least-cost, technically adequate supplier; and a minimum of bothersome requirements changes. Many of our current acquisition institutions – regulations, specifications, standards, contract types, award fee structures, reviews and audits – are optimized around this procurement model.

Such institutions have been the bane of many projects attempting to deliver successful systems in a world of emerging requirements and rapid change. The project people may put together good technical and management strategies to do concurrent problem and solution definition, teambuilding, and mutual-learning prototypes and options analysis. Then they find that their progress payments and award fees involve early delivery of complete functional and performance specifications. Given the choice between following their original strategies and getting paid, they proceed to marry themselves in haste to a set of premature requirements then find themselves repenting at leisure for the rest of the project (if any leisure time is available).

Build-to-specification contract mechanisms still have their place, but it is just for the stabilized increment development. If such mechanisms are applied to the agile rebaselining teams, then frustration and chaos ensues. What is needed for the three-team approach are separate contracting mechanisms for the functions, under an overall contract structure, enabling them to be synchronized and rebalanced across the life cycle. Also needed are source-selection mechanisms more likely to choose the most competent supplier, using such approaches as competitive exercises to develop representative system artifacts using the people, products, processes, methods, and tools in the offeror's proposal.

A good transitional role model is the Command Center Processing and Display-Replacement (CCPDS-R) project described in [11]. Its U.S. Air Force customer and TRW contractor (selected using a competitive exercise such as the one described earlier) reinterpreted the traditional defense regulations, specifications, and standards. They held a preliminary design review: This was not a PowerPoint show at month four, but a fully validated architecture and demonstration of the working, high-risk user interface and networking capabilities at month 14. The resulting system delivery, including more than one million lines of software source code, exceeded customer expectations within budget and schedule.

Other good acquisition approaches are the Scandinavian Participatory Design approach [12], Checkland's Soft Systems Methodology [13], lean acquisition and development processes [14], and Shared Destiny-related contracting mechanisms and award fee structures [15, 16]. These all reflect the treatment of acquisition using a C2ISR metaphor rather than a purchasing-agent metaphor.

Model Experience to Date and Conclusions

The scalable spiral model has been evolving with experience and has not yet been fully implemented on a large, completed project. However, its principles and practices build on many successful project experiences and unsuccessful project lessons learned. Specific examples of projects that have successfully balanced agile and plan-driven methods are the agile-based ThoughtWorks lease management project [17] and the plan-based CCPDS-R project [11]. More generally, J. Collins' book, "Good to Great" [18] identifies 11 companies with exceptional performance records as having successfully transformed themselves into having both a strong ethic of entrepreneurship and a strong culture of discipline.

The use of concurrent IV&V teams has been successfully practiced and evolved since the 1970s [19]. More recent successful continuous IV&V practices include the continuous build practices at Microsoft [20] and in agile methods [21]. Proactive investments in agile next-increment teams are successfully used in exploiting disruptive technologies at companies such as Hewlett Packard (HP), Seagate, and Johnson and Johnson [22]; and in practicing open innovation in companies such as HP, IBM, Intel, and Lucent [23]. Successful use of the anchor point milestones and evolutionary development using the Rational Unified Process [16] and the WinWin Spiral model [24] has been experienced on numerous small, medium, and large software projects and on hardware projects at such companies as Xerox and Johnson and Johnson. Partial implementations of the model are also providing improvement and are being evolved on the large-scale U.S. Army Future Combat Systems program, large space systems, and commercial supply chain systems [8].

Experience to date indicates that the three teams' activities are not as neatly orthogonal as they look in Figures 1 and 3. Feedback on development shortfalls from the IV&V team either requires a response from the development team (early fixes will be less disruptive and expensive than later fixes), or deferral to a later increment, adding work and coordination by the agile team. The agile team's analysis and prototypes addressing how to accommodate changes and deferred capabilities need to draw on the experience and expertise of the plan-driven development team, requiring some additional development team resources and calendar time. Additional challenges arise if different versions of each increment are going to be deployed in different ways into different environments. The model has sufficient degrees of freedom to address such challenges, but they need to be planned within the project's schedule and budget.

In working with our commercial and aerospace affiliates on how they can best evolve to succeed as 21st century enterprises, we have found several 20th century process-related institutions that need to be significantly rethought and reworked to contribute to success. Two key leading areas for SISOS development that need rethinking are acquisition practices and human relations [10]. Other institutions that also need rethinking and rework are continuous process improvement (repeatability and optimization around the past versus adaptability and optimization around the future), supplier management (adversarial win-lose versus team-oriented win-win), internal research and development strategies (core capability research plus external technology experimentation versus full-spectrum self-invention), and enterprise integration (not-invented-here stovepipes versus enterprise-wide learning and sharing).◆

References

1. International Organization for Standardization. Information Technology – Software Life Cycle Processes. ISO/IEC 12207: 1995. Geneva, Switzerland: ISO, 1995.
2. International Organization for Standardization. Systems Engineering – System Life Cycle Processes. ISO/IEC 15288: 2002. Geneva, Switzerland: ISO, 2002.
3. Zachman, J. "A Framework for Information Systems Architecture." IBM Systems Journal 26.3 (1987): 276-292.
4. Putman, J. Architecting With RM-ODP. Prentice Hall, 2001.
5. Federal Chief Information Officer Council. A Practical Guide to Federal Enterprise Architecture Vers. 1.0. Washington, D.C.: FCIO, Feb. 2001.
6. Harned, D., and J. Lundquist. "What Transformation Means for the

Defense Industry." The McKinsey Quarterly 3 Nov. 2003: 57-63.
7. Lane, J., and R. Valerdi. "Synthesizing System-of-Systems Concepts for Use in Cost Estimation." IEEE SMC, 2005.
8. Boehm, B., A.W. Brown, V. Basili, and R. Turner. "Spiral Acquisition of Software-Intensive Systems of Systems." CrossTalk May 2004: 4-9 <www.stsc.hill.af.mil/crosstalk/2004/05/0405boehm.html>.
9. U.S. Air Force-Scientific Advisory Board. Information Architectures Study. Washington, D.C.: U.S. Air Force, 1994.
10. Boehm, B. "Some Future Trends and Implications for Systems and Software Engineering Processes." USC-CSE-TR-2005-507. Los Angeles, CA: University of Southern California, 2005.
11. Royce, W.E. Software Project Management. Addison-Wesley, 1998.
12. Ehn, Pelle. Work-Oriented Design of Computer Artifacts. Lawrence Earlbaum Assoc., 1990.
13. Checkland, P. Systems Thinking, Systems Practice. 2nd ed. Wiley, 1999.
14. Womack, J., and D. Jones. Lean Thinking: Banish Waste and Create Wealth in Your Corporation. Simon & Schuster, 1996.
15. Deck, M., M. Strom, and K. Schwartz. "The Emerging Model of Co-Development." Research Technology Management Dec. 2001.
16. Rational, Inc. Driving Better Business With Better Software Economics. Rational Software Corp., 2001.
17. Elssamadisy, A., and G. Schalliol. Recognizing and Responding to 'Bad Smells' in Extreme Programming. Proc. of the 24th International Conference on Software Engineering, Orlando, FL, May 2002: 617-622.
18. Collins, J. Good to Great. Harper Collins, 2001.
19. Rubey, J.R., J.A. Dana, and P.W. Biché. "Quantitative Aspects of Software Validation." IEEE Transactions on Software Engineering June 1975: 150-155.
20. Cusumano, M., and R. Selby. Microsoft Secrets. Harper Collins, 1996.
21. Beck, K. Extreme Programming Explained. Addison-Wesley, 1999.
22. Christensen, C. The Innovator's Dilemma (Harper Business Essentials). Harper Collins, 2000.
23. Chesbrough, H. Open Innovation: The New Imperative for Creating and Profiting from Technology. Harvard Business School Press, 2003.
24. Boehm, B., A. Egyed, J. Kwan, D. Port, A. Shah, and R. Madachy. "Using the WinWin Spiral Model: A Case Study." IEEE Computer July 1998: 33-44.

About the Authors

Barry Boehm, Ph.D., is the TRW professor of software engineering and director of the Center for Software Engineering at the University of Southern California. He was previously in technical and management positions at General Dynamics, Rand Corp., TRW, and the Defense Advanced Research Projects Agency, where he managed the acquisition of more than $1 billion worth of advanced information technology systems. Boehm originated the spiral model, the Constructive Cost Model, and the stakeholder win-win approach to software management and requirements negotiation.

University of Southern California
Center for Software Engineering
941 W 37th PL
SAL RM 328
Los Angeles, CA 90089-0781
E-mail: boehm@usc.edu

Jo Ann Lane is currently a research assistant supporting software engineering and system-of-systems research activities at the University of Southern California's Center for Software Engineering. In this capacity, she is currently working on a cost model to estimate the effort associated with system-of-systems architecture definition and integration. Prior to this, she was a key technical member of Science Applications International Corporation's Software and Systems Integration Group. She has over 28 years of experience in the development of software-intensive systems.

University of Southern California
Center for Software Engineering
941 W 37th PL
SAL Room 328
Los Angeles, CA 90089-0781
E-mail: jolane@usc.edu

Coming Events

June 17-21
33rd Annual International Symposium on Computer Architecture (ISCA 2006)
Boston, MA
www.ece.neu.edu/conf/isca2006

June 19-22
CISC 2006 Combat Identification Systems Conference
Orlando, FL
www.usasymposium.com/cisc

June 25-30
18th Annual FIRST (Forum of Incident Response and Security Teams) Conference on Computer Security and Incident Handling
Baltimore, MD
www.first.org/conference/2006

June 26-29
SERP 2006
International Conference on Software Engineering Research and Practice
Las Vegas, NV
http://people.cs.und.edu/~reza/SERP06.html

June 26-30
2006 Better Software Conference and Expo
Las Vegas, NV
www.sqe.com/bettersoftwareconf.org

June 27-29
CMSE Europe
Components for Military and Space Electronics Workshop and Conference
Portsmouth, U.K.
www.cmse-eur.com

April 16-19, 2007
2007 Systems and Software Technology Conference
Salt Lake City, UT
www.sstc-online.org

Tackling the Cost Challenges of System of Systems

Arlene F. Minkiewicz
PRICE Systems

Monday, 1 May 2006
Track 1: 3:55 – 4:40 p.m.
Ballroom A

The Department of Defense and its contractors are currently facing unprecedented challenges in planning projects involving groups of systems integrated into one large system of systems (SOS). These challenges are intensified by the fact that these systems tend to be heavily software-dependent. Often planners must decide which configuration of platforms best meets mission needs with respect to affordability, performance, and risk in the very early stages of a project from top-level requirements. This article presents research of the cost issues associated with delivery of SOS capabilities. It starts with a discussion on what an SOS is and areas where SOS projects vary from typical system development and deployment. New and expanded contractor roles and activities are presented, highlighting how these drive cost differences from traditional system projects. Guidelines are provided for performing high-level analysis of SOS costs to enable decision makers to perform trade-offs between various configurations in order to pursue the most affordable solution that will meet mission needs.

The Department of Defense (DoD) has migrated from a platform-based acquisition strategy to one focused on delivering capabilities. Instead of delivering a fighter aircraft or an unmanned air vehicle, contractors are now being asked to deliver the right collection of hardware and software to meet specific wartime challenges. This means that much of the burden associated with conceptualizing, architecting, integrating, implementing, and deploying complex capabilities into the field has shifted from desks in the Pentagon to desks at Lockheed Martin, Boeing, Rockwell, and other large aerospace and defense contractors.

In "The Army's Future Combat Systems' [FCS] Features, Risks and Alternatives," the Government Accounting Office states the challenge as:

> …14 major weapons systems or platforms have to be designed and integrated simultaneously and within strict size and weight limitations in less time than is typically taken to develop, demonstrate, and field a single system. At least 53 technologies that are considered critical to achieving critical performance capabilities will need to be matured and integrated into the system of systems. And the development, demonstration, and production of as many as 157 complementary systems will need to be synchronized with FCS content and schedule. [1]

The planning, management, and execution of such projects will require changes in the way organizations do business. This article reports on ongoing research into the cost challenges associated with planning and executing a system of systems (SOS) project. Because of the relatively immature nature of this acquisition strategy, there is not nearly enough hard data to establish statistically significant cost-estimating relationships. The conclusions drawn to date are based on what we know about the cost of system engineering and project management activities in more traditional component system projects augmented with research on the added factors that drive complexities at the SOS level.

The article begins with a discussion of what an SOS is and how projects that deliver SOS differ from those projects delivering stand-alone systems. Following this is a discussion of the new and expanded roles and activities associated with SOS that highlight increased involvement of system engineering resources. The focus then shifts to cost drivers for delivering the SOS capability that ties together and optimizes contributions from the many component systems. The article concludes with some guidelines for using these cost drivers to perform top-level analysis and trade-offs focused on delivering the most affordable solution that will satisfy mission needs.

Related Research

Extensive research has been conducted on many aspects of SOS by the DoD, academic institutions, and industry. Earlier research focused mainly on requirements, architecture, test and evaluation, and project management [2, 3, 4, 5, 6, 7, 8]. As time goes on and the industry gets a better handle on the technological and management complexities of SOS delivery, the research expands from a focus on the right way to solve the problem to a focus on the right way to solve the problem *affordably*. In the forefront of this cost-focused research is the University of Southern California's Center for Software Engineering [9], the Defense Acquisition University [10], Carnegie Mellon's Software Engineering Institute [11], and Cranfield University [12].

What Is an SOS?

An SOS is a configuration of component systems that are independently useful but synergistically superior when acting in concert. In other words, it represents a collection of systems whose capabilities, when acting together, are greater than the sum of the capabilities of each system acting alone.

According to Mair [13], an SOS must have most, if not all, of the following characteristics:

- Operational independence of component systems.
- Managerial independence of component systems.
- Geographical distribution.
- Emergent behavior.
- Evolutionary development processes.

For the purposes of this research, this definition has been expanded to explicitly state that there be a network-centric focus that enables these systems to communicate effectively and efficiently.

Today, there are many platforms deployed throughout the battlefield with limited means of communication. This becomes increasingly problematic as multiple services are deployed on a single mission as there is no consistent means for the Army to communicate with the Navy or the Navy to communicate with the Air Force. Inconsistent and unpredictable means of communication across the battlefield often results in unacceptable time from detection of a threat to engagement. This can ultimately endanger the lives of our service men and women.

One example of an SOS that the Army is currently envisioning is the Warfighter Information Network-Tactical (WIN-T), which is a communication system designed for reliable, secure, and seamless video, data, imagery, and voice services to enable decisive, real-time combat actions. This SOS promises full, two-way communication between platforms and across services, making it possible for information to be shared and processed in time to make a real difference in the outcome. The cloud is being lifted from the battlefield!

How Different Are SOS Projects?

How much different is a project intended to deliver an SOS capability from a project that delivers an individual platform such as an aircraft or a submarine? Each case presents a set of customer requirements that need to be elicited, understood, and maintained. Based on these requirements, a solution is crafted, implemented, integrated, tested, verified, deployed, and maintained. At this level, the two projects are similar in many ways. Dig a little deeper and differences begin to emerge. The differences fall into several categories: acquisition strategy, software, hardware, and overall complexity

The SOS acquisition strategy is capability-based rather than platform-based. For example, the customer presents a contractor with a set of capabilities to satisfy particular battlefield requirements. The contractor then needs to determine the right mix of platforms, the sources of those platforms, where existing technology is adequate, and where invention is required. Once those questions are answered, the contractor must decide how best to integrate all the pieces to satisfy the initial requirements. This capability-based strategy leads to a project with many diverse stakeholders. Besides the contractor selected as the lead system integrator (LSI), other stakeholders that may be involved include representatives from multiple services, Defense Advanced Research Projects Agency, prime contractor(s) responsible for supplying component systems as well as their subcontractors. Each of these stakeholders brings to the table different motivations, priorities, values, and business practices – each brings new people management issues to the project.

Software is an important part of most projects delivered to DoD customers. In addition to satisfying the requirements necessary to function independently, each of the component systems needs to support the interoperability required to function as a part of the entire SOS solution. Much of this interoperability will be supplied through the software resident in the component systems. This requirement for interoperability dictates that well-specified and applied communication protocols are a key success factor when deploying an SOS. Standards are crucial, especially for the software interfaces. Additionally, because of the need to deliver large amounts of capability in shorter and shorter timeframes, the importance of commercial off-the-shelf (COTS) software in SOS projects continues to grow.

With platform-based acquisitions, the customer generally has a fairly complete understanding of the requirements early on in the project with a limited amount of requirements growth once the project commences. Because of the large scale and long-term nature of capability-based acquisitions, the requirements tend to emerge over time with changes in governments, policies, and world situations. Because requirements are emergent, planning and execution of both hardware and software contributions to the SOS project are impacted.

SOS projects are also affected by the fact that the hardware components being used are of varying ages and technologies. In some cases, an existing hardware platform is being modified or upgraded to meet increased needs of operating in an SOS environment, while in other instances brand new equipment with state-of-the-art technologies is being developed. SOS project teams need to deal with components that span the spectrum from the high-tech, but relatively untested to the low-tech, tried-and-true technologies and equipment.

Basically, a project to deliver an SOS capability is similar in nature to a project intended to deliver a specific platform except that overall project complexity may be increased substantially. These complexities grow from capability-based acquisition strategies, increased number of stakeholders, increased overall cost (and the corresponding increased political pressure), emergent requirements, interoperability, and equipment in all stages from infancy to near retirement.

New and Expanded Roles and Activities

Understanding the manifestation of these increased complexities on a project is the first step to determining how the planning and control of an SOS project differs from that of a project that delivers one of the component systems. One of the biggest and most obvious differences in the project team is the existence of an LSI. The LSI is the contractor tasked with the delivery of the SOS that will deliver the capabilities the DoD customer is looking for. The LSI can be thought of as the *super prime* or the *prime of prime* contractors. He or she is responsible for managing all the other primes and contractors and ultimately for fielding the required capabilities. The main areas of focus for the LSI include:

- Requirements analysis for the SOS.
- Design of SOS architecture.
- Evaluation, selection, and acquisition of component systems.
- Integration and test of the SOS.
- Modeling and simulation.
- Risk analysis, avoidance, and mitigation.
- Overall program management for the SOS.

One of the primary jobs of the LSI is completing the system engineering tasks at the SOS level.

Focus on System Engineering

The following is according to the "Encyclopedia Britannica":

> … system engineering is a technique of using knowledge from various branches of engineering and science to introduce technological innovations into the planning and development stages of systems. Systems engineering is not as much a branch of engineering as it is a technique for applying knowledge from other branches of engineering and disciplines of science in an effective combination. [14]

System engineering as a discipline first emerged during World War II as

technology improvements collided with the need for more complex systems on the battlefield. As systems grew in complexity, it became apparent that it was necessary for there to be an engineering presence well versed in many engineering and science disciplines to lend an understanding of the entire problem a system needed to solve. To quote Admiral Grace Hopper, "Life was simple before World War II. After that, we had systems [15]."

With this top-level view, the system engineers were able to grasp how best to optimize emerging technologies to address the specific complexities of a problem. Where an electrical engineer would concoct a solution focused on the latest electronic devices and a software engineer would develop the best software solution, the system engineer knows enough about both disciplines to craft a solution that gets the best overall value from technology. Additionally, the system engineer has the proper understanding of the entire system to perform validation and verification upon completion, ensuring that all component pieces work together as required.

Today, a new level of complexity has been added with the emerging need for SOS, and once again the diverse expertise of the system engineers is required to overcome this complexity. System engineers need to comprehend the big picture problem(s) whose solution is to be provided by the SOS. They need to break these requirements down into the hardware platforms and software pieces that best deliver the desired capability, and they need to have proper insight into the development, production, and deployment of the component systems to ensure not only that they will meet their independent requirements, but also that they will be designed and implemented to properly satisfy the interoperability and interface requirements of the SOS. It is the task of the system engineers to verify and validate that the component systems, when acting in concert with other component systems, do indeed deliver the necessary capabilities.

Cost Considerations of SOS Projects

An SOS is a collection of existing, upgraded, and new systems that are required to work together to accomplish specific objectives. Clearly the costs of developing and acquiring component systems is one important cost consideration, but since estimating system costs is a fairly mature discipline, this article focuses on the additional costs associated with the delivery of capabilities made possible when a configuration of such systems works as a system.

Mastering the cost questions in an SOS project first requires establishing a link between the increased complexities and the participation of system engineers in the project. A traditional parametric estimating methodology for hardware or software systems relies on a quantification of the size and complexity of the system being developed. Size is driven by weight for hardware and source lines of code or function points for software. Project, process, and organizational factors drive complexity. Assigning a size and complexity to an SOS is a bit trickier. Traditional size measures alone are not adequate for estimating the size of system engineering tasks and with many participating organizations, the process and organizational factors can vary substantially within the project team.

"Mastering the cost questions in an SOS project first requires establishing a link between the increased complexities and the participation of system engineers in the project."

Tricky or not, being able to properly size the SOS part of a project is crucial to successfully determining what it will cost and how long it will take to deliver. It is also a crucial step in being able to make trade-offs in order to deliver a solution that not only meets requirements, but also satisfies affordability constraints. As with all estimating, the challenge in sizing an SOS is being able to translate what is known early on in the project into information that represents useful project characteristics as the project evolves.

Toward this end, our research indicates that the number of unique interface protocols and the number of different component systems are the two best factors for determining the size of the SOS effort. In an SOS project, it is the LSI's job to define and design the infrastructure that will facilitate communication among the many component systems. The number of unique interface protocols is clearly a good start for determining problem space size. Augmentation of this number with the number of component systems that will be designed for or adapted to operate within this infrastructure provides an even better proxy for the size of the solution. This conclusion is consistent with the research done at the University of Southern California's Center for Software Engineering on the Constructive System of System Integration Model [9].

The number of unique interface protocols drives the size of the integration and test effort. Our experience is that the effort for integration and test within a typical system ranges between 5 percent and 40 percent of the entire development effort of the system as the number of interfaces goes from few to many; this effect would be exaggerated in an SOS as complexity of the overall integration problem is greater. As the number of component systems increases, integration efforts increase in a non-linear fashion as a result of the diseconomy of scale brought on by project complexity. Additionally, the number of components will influence management and oversight costs in the form of added people and communication issues.

Size, of course, is only part of the puzzle. Multiple SOS within the same size range will only fall into the same cost range as a coincidence. For the sake of this discussion, consider the simplistic cost model that applies an exponent and a coefficient to a project size. In this context, the size is as described and the exponent and coefficient are determined by factors that determine project complexity. As such, it is necessary to assign relative complexity values to the various configurations. There are many factors that have a potential impact on complexity, some that are obvious early on in the project, and others that will emerge throughout the project life cycle. The ones that are available or predictable early on in the project and that appear to have the most significant impact on the amount of effort required for the SOS tasks include those in the following sections.

Number of Operational Scenarios

An operational scenario refers to a particular capability instance for some set of the component systems of the SOS. For example, the Coast Guard's Integrated

Deepwater System needs to include capability that can react to a terrorist threat, a person lost at sea, or a drug-smuggling operation. The number of operational scenarios impacts the coefficient in the cost equation discussed earlier as additional scenarios result in more time for requirements, design, and modeling and simulation. Depending on the similarities of the scenarios, the impacts to these activity's costs should represent increases between 10 percent and 50 percent.

Required Level for Acceptance of Key Performance Parameters

Key performance parameters associated with an SOS include things like detection effectiveness, survivability, and lethality. This factor could have substantial impact on both the coefficient and the exponent in the simple cost model mentioned earlier. System engineering activities associated with the SOS could double or triple, or more as the detection effectiveness expectations move from available technology to state of the art. Use of immature technology on the Joint Tactical Radio System Program was cited as one of the main reasons for a $458 million development cost increase [16].

Number of Suppliers and Stakeholders

The number of players involved in an SOS project can increase the complexity and cost significantly. On a typical system project, people and communication issues can increase the cost of project management and oversight activities by as much as 60 percent. This effect can increase dramatically as the relatively well-known confines of the typical system are replaced with the much more expansive and undefined constraints on an SOS project.

Integration Complexity

Integration complexity is a quantification of the amount of integration each component is expected to require with the rest of the SOS. An SOS that requires highly complex integrations within and among each of its component systems could potentially see the integration and test activity costs increase an order of magnitude from an SOS where all of the integrations are simple, well-defined tasks.

Stability and Readiness of Components

As mentioned earlier, the technical immaturity of components can substantially impact system engineering tasks. Additionally, immature components can impact the overall schedule and cost of the SOS, since integration and test activities for various capabilities will be delayed until all required components are available. The WIN-T program was originally planned to deliver technologies not expected to mature until after production started. Such a strategy is guaranteed to lead to costly schedule delays.

Amount of COTS Capability

COTS components generally require modification, integration, and test, as well as compromise on SOS requirements. When looking at the overall cost for an SOS, off-the-shelf components should decrease the cost compared to newly developed components. From the perspective of the LSI, however, they represent an increase of system engineering effort associated with requirements, design and integration, and test. This cost increase can be quite modest if the components and vendors are chosen wisely, but it could double the costs of these activities if poor choices are made.

"Although there is an upfront investment in getting the architectures right and standardizing communication protocols, the payoff is significant during delivery of the initial operating concept and throughout the life of the SOS."

Affordable SOS

When crafting a solution to deliver an SOS capability, there are things the LSI can do to ensure that it not only meets all performance requirements, but does so within affordability constraints. All possible solutions should be focused on the specified constraints for stated key performance parameters (KPPs). No solution should be presented that does not satisfy these constraints. Component systems that drive performance substantially above specified performance in these areas should be carefully scrutinized as well. All possible solutions should first be validated to ensure that they successfully address all KPPs and support all operational scenarios.

Care should be taken to utilize as many existing component systems as possible rather than developing new ones. When new component systems must be developed to deliver some currently non-existing capability or degree of performance, it is important to get the most from the technology investment. Attempts should be made to incorporate as much capability as practical into the new development to reduce the number of different component systems. Increases in complexity associated with technology readiness and component stability may be offset by size decreases if the number of required component systems can be reduced. At the same time, care should be taken to ensure that expectations for technology do not exceed practical limits on innovation imposed by schedule constraints on the program.

Well-thought-out architecture with simple communication protocols that meet many different needs will reduce the size of the SOS solution space. Although there is an up-front investment in getting the architectures right and standardizing communication protocols, the payoff is significant during delivery of the initial operating concept and throughout the life of the SOS. Emerging requirements will result in the addition of new component systems that must communicate with existing components.

The use of COTS hardware and software is a practical and necessary approach to accomplish the delivery of SOS capabilities in required timeframes. When possible, the same vendor should be considered for multiple components, parts, or software products. This reduces the number of vendors involved in the project, eases the effort to integrate between components, and could possibly result in favorable purchasing agreements based on bulk. Integration complexity can also be reduced through simple standards that are strictly enforced, effective risk management techniques focused on early identification and mitigation, and ongoing integration efforts.

Conclusion

Today, SOS solutions are replacing the existing post-World War II systems as

the next generation of complex solutions supplied by contractors to the DoD. SOS projects require contractors to deliver capabilities rather than standalone systems. Contractors are left to decide on and acquire component systems, determine the best configuration for these component systems to achieve the required capabilities, and develop the best plan for interoperability among the component systems.

While there are some ways in which an SOS project is similar to a project that delivers a component system, there are many ways in which the two types of projects differ. Understanding these differences and how they affect the cost and effort associated with a project is crucial to proper planning and execution of an SOS project. A crucial difference is the requirement for increased involvement of system engineering resources throughout the life cycle of the SOS project. System engineers are involved in requirements elicitation and management, architecture decisions, test and evaluation, verification and validation, and technical oversight for the SOS project.

Cost drivers for an SOS fall into two categories: those that define the size of the system engineering tasks, and those that drive the complexity of the engineering and management tasks. Because the notion of capability-based acquisitions is still relatively immature, there is not the preponderance of data required to develop good, strong, cost-estimating relationships for SOS project activities. Despite this, it is possible – and necessary – to begin estimating these projects today by incorporating estimating knowledge gained through years of system development augmented with information about the additional factors that influence SOS project size and complexities. Future directions for this research involve collecting data from evolving SOS projects as they reach milestones and use this data to refine, update, or replace cost-estimating relationships.◆

References

1. Francis, Paul L. The Army's Future Combat System's Features, Risks and Alternatives. Testimony before the Subcommittee on Tactical Air and Land Forces. Committee on Armed Services. House of Representatives, GAO-04-635T 1 Apr. 2004 <www.gao.gov/new.items/d04635t.pdf>.
2. Lamartin, Glenn. "The Role of T&E in the Systems Engineering Process." National Defense Industrial Association System Engineering Conference, 17 Aug. 2004 <www. acq.osd.mil/ds/se/speeches.htm>.
3. Hooks, Ivy. "Managing Requirements for a System of Systems." CrossTalk Aug. 2004 <www.stsc.hill.af.mil/crosstalk/2004/08/index.html>.
4. Krone, Roger. "Managing a Complex System-of-Systems." President's Commission on Moon, Mars and Beyond, 4 May 2004 <www.govinfo.library.unt.edu/moontomars/news/docs.asp>.
5. Martin, James N. "Modeling and Architecture Considerations for Systems of Systems." 2004 Systems and Software Technology Conference, Salt Lake City, UT, 21 Apr. 2004.
6. Carney, D., and P. Oberndorf. "Integration and Interoperability Models for Systems of Systems." 2004 Systems and Software Technology Conference, Salt Lake City, UT, 1 Apr. 2004.
7. Crossley, William A. "System of Systems: An Introduction of Purdue University Schools of Engineering's Signature Area." Engineering Systems Symposium, MIT Engineering Systems Division, Mar. 2004.
8. Conrow, Edmund H. "Risk Management for Systems of Systems." CrossTalk Feb. 2005 <www.stsc.hill.af.mil/crosstalk/2005/02/0502conrow.html>.
9. Lane, Jo Ann. "Factors Influencing System-of-Systems Architecting and Integration Costs." University of Southern California's Center for Software Engineering <www.stevens-tech.edu/cser/authors/46.pdf>.
10. Flowe, R., and M. Spurlock. "Systems of Systems Research Project Overview." Office of the Secretary of Defense Program Analysis and Evaluation, Mar. 2004 <http://acc.dau.mil/simplify/ev_en.php >.
11. Zubrow, Dave. "System of Systems Integration Cost Driver Research." 37th Annual DoD Cost Analysis Symposium, Feb. 2004.
12. Adcock, Rick. "Principles and Practices of Systems Engineering." INCOSE UK Chapter Library, Nov. 2001 <www.incose.org.uk/library.htm>.
13. Mair, M.W. "Architecting Principles for System-of-Systems." Systems Engineer 1.4 (1998).
14. Encyclopedia Britannica Online <www.britannica.com>.
15. Wikipedia <http://en.wikipedia.org/wiki/system.engineering>.
16. Francis, Paul L. Defense Acquisitions, Future Combat Systems Challenges and Prospects for Success. Testimony Before the Sub-committee on Airland. Committee on Armed Services. U.S. Senate, Government Accounting Office, 16 Mar. 2005 <www.gao.gov>.

About the Author

Arlene F. Minkiewicz is chief scientist of the Cost Research Department at PRICE Systems. She is responsible for the research and analysis necessary to keep the suite of PRICE estimating products responsive to current cost trends. In her 20-year tenure with PRICE, Minkiewicz has researched and developed the software cost estimating relationships that were the cornerstone for PRICE's commercial software cost estimating model, ForeSight, and invented the Cost Estimating Wizards originally used in ForeSight that walk the user through a series of high-level questions to produce a quick cost analysis. As part of this effort she has invented a sizing measurement paradigm for object-oriented analysis and design that allows estimators a more efficient and effective way to estimate software size. She recently received awards from the International Society of Parametric Analysts and the Society of Cost Estimating and Analysis for her white paper "The Real Cost of COTS." Minkiewicz contributed to a new parametric cost estimating book with the Consortium for Advanced Manufacturing International called "The Closed Loop: Implementing Activity-Based Planning and Budgeting," and she frequently publishes articles on software estimation and measurement. She has also been a contributing author for several books on software measurement and speaks frequently on this topic at numerous conferences.

PRICE Systems
17000 Commerce PKWY STE A
Mt. Laurel, NJ 08054
Phone: (856) 608-7222
Fax: (856) 608-7247
E-mail: arlene.minkiewicz@pricesystems.com

Building Multilevel Secure Web Services-Based Components for the Global Information Grid

Dylan McNamee
Galois Connections, Inc.

CDR Scott Heller
Program Executive Office C4I and Space

Dave Huff
Fleet Numerical Meteorology and Oceanography Center

Thursday, 4 May 2006
Track 3: 9:55 – 10:40 a.m.
Ballroom C

A consensus is growing that the Department of Defense's vision of a future Global Information Grid will be built using architecture that takes advantage of Web services and uses standard Internet protocols, interchangeable components, and commercially available hardware and software wherever possible. This article describes the features and architecture of two systems: the Trusted Services Engine and the Multilevel Document Collaboration Server, including their use of a separation kernel with multiple independent levels of security, the design and assurance architecture of the cross-domain block-access controller, and the composition architecture that extends the inter-level isolation property from the block access controller outward through complex services.

The Global Information Grid (GIG) is the overall architecture intended to replace current stovepipe information systems. A consensus is growing that the Department of Defense's vision of this future GIG will use an architecture that takes advantage of Web services and uses standard Internet protocols, interchangeable components, and commercially available hardware and software wherever possible. By adopting modern standards-based protocols, the GIG will enhance current capability by enabling people and components to work together dynamically with integrated data.

Protocols such as Hypertext Transfer Protocol, eXtensible Markup Language (XML), Web-based Distributed Authoring and Versioning (WebDAV), Really Simple Syndication, and Lightweight Directory Access Protocol allow the GIG to be made of off-the-shelf components where appropriate. Where custom components are required, pervasive use of these protocols preserves the component-based architecture of the GIG, thus protecting the architecture from developing into a stovepipe system.

Many of these components and protocols are mature and well understood, but they were not designed with security as the paramount consideration. Securing the GIG is therefore a significant challenge. Particularly critical is securing its cross-domain services. For these, the GIG itself must somehow enforce separate levels of security.

Today, physical isolation enforces separation, though other technologies such as cryptography may someday be used. Such separation allows the use of commercial components as single-level components not responsible for cross-domain security concerns. However, for the GIG to realize its potential, some components must enable secure *cross-domain* data access. Clearly such components, while they must conform to commercial protocols, must be developed to *higher than* commercial standards.

> *"In particular, the greater security risks associated with cross-domain components – as compared to single-level, commercial solutions – require a correspondingly higher level of trust."*

This article, which describes such a component, has three main parts:

1. We describe the security and assurance attributes required of a cross-domain component of the GIG.
2. We describe the architecture and technologies we are using to achieve these attributes in the Trusted Services Engine (TSE), a network-enabled file store with integrated read-down across security domains.
3. We conclude by describing a system built on the TSE, the Multilevel Document Collaboration Server, to enable cross-domain collaboration *within* documents – an example of using simple cross-domain components to build more complex cross-domain systems using only standard protocols and APIs.

This article describes the features and architecture of both systems:

- The design and assurance architecture of the cross-domain *block access controller* (BAC).
- The use of a Multiple Independent Levels of Security (MILS) separation kernel.
- The composition architecture that extends the cross-domain isolation property from the MILS separation kernel to the BAC and outward through complex services.

This article is focused toward a technical audience familiar with Web services.

Assurance Requirements for Cross-Domain GIG Components

The nature and mission of the GIG makes it a prime target for trained, well-funded, and resourceful adversaries. The threats posed by such adversaries, coupled with the value of the information on the GIG, require us to show that the GIG components are robust in the face of these threats. In particular, the greater security risks associated with cross-domain components – as compared to single-level, commercial solutions – require a correspondingly higher level of trust. The process of generating and evaluating evidence of trustworthiness is known as *assurance*, the most difficult aspect of security engineering.

Two processes in the defense and intelligence communities support each other to generate assurance evidence for a GIG component: evaluation and certification. *Evaluation* is the process of validating securi-

ty claims for a particular component. For example, the Common Criteria is an international standard for specifying claims of system security functionality and generating assurance that these claims are satisfied. We have determined that the cross-domain components we are building will need to meet the requirements for Common Criteria's Evaluation Assurance Level 6 or 7 [1].

Certification focuses on verifying that a component can be securely deployed at a particular site. Certification is best represented by such processes as Secret and Below Interoperability, and Top Secret and Below Interoperability. What these processes have in common is a way to tailor requirements for evaluation or certification of the following:

- Sensitivity of the data that the component handles.
- Severity of the threats it must withstand.

For example, under Director of Central Intelligence Directive 6/3, a cross-domain component that needs to demonstrate high assurance with respect to confidentiality must satisfy Protection Level 4 or 5 assurance requirements. Evaluating or certifying a component to one of those standards requires an extensive investment in time and resources. But given the responsibilities of a cross-domain component of the GIG, high assurance is a must.

Architecture for a High-Assurance GIG Component

The TSE, a government off-the-shelf software development project funded by the Space and Naval Warfare Systems Command (SPAWAR) and National Security Agency, is a network-enabled file store with integrated read-down across security domains. The TSE provides the file store using the standard WebDAV protocol. It has a separate hardware network interface for each network security level and a separate file store for data at each level.

The TSE enforces the Bell-LaPadula policy of information flow [2], in which users on each network can read from their own level and below, but can write only to their own level. For example, when one security level dominates another (for example, TOP SECRET dominates SECRET), the TSE allows *read-down* – the ability for users at a higher level to access data from a lower level, but not vice-versa. All levels share a single name space, but views of that name space differ according to the network security level accessing the TSE.

Read-down eliminates the need for low-security data to be explicitly copied for users at high security. The single name space combined with read-down makes a wide range of applications and user workflows easier, more dynamic, and less error-prone than existing solutions.

Developing, certifying, and evaluating a high assurance cross-domain component such as the TSE at acceptable cost requires a fundamentally different architecture from that of typical, single-level components. Our approach is the following: Use as few high-assurance components as possible, each with a single purpose, to keep it small and simple, allowing it to be analyzed formally. But security is a property of a whole system, not just a component. Appropriate composition techniques can extend the security properties of the trusted computing base outward to the rest of the system.

"The problem is caused by a read-down – a user on a high-level network can read files from a lower level while a user on the low-level network changes those files."

The TSE's trusted computing base consists of the minimum number of components: one. TSE functionality is decomposed into a set of single-level components and only one cross-domain component. The underlying MILS separation kernel separates components at different security levels. Each network security level has a set of clients, an authentication service, and an integrity checker (see Figure 1). Within the TSE, each network level has its own network interface card, hard drive, and software stack implementing the TSE's networking, WebDAV, and file system services.

The TSE's only cross-domain component, the BAC, mediates all access between the TSE and each level's disks.

How can these components be assembled to provide secure, cross-domain services?

1. The base must be secure before building on it. We must first establish the isolation properties of the cross-domain component.
2. We can then extend these properties to physically separate networks by mapping the software components to separate partitions in the separation kernel.
3. Finally, the separation kernel is configured to permit communication only between appropriate components.

The Cross-Domain Component

Together with the separation kernel, the BAC is responsible for isolating each level in the TSE. It is, therefore, the component that needs to be evaluated and certified to the highest levels of assurance. The BAC's functions are the following:

- Mediate all disk block access.
- Connect single-level disks and partitions.
- Write blocks to the same level.
- Read blocks from the same or lower levels.

The keys to BAC security are that it has a well-defined job and is constructed from very few lines of code. The current version of the BAC is 780 lines of *C* code. To ensure that the BAC implements the required attributes, we do the following:

1. Develop a formal model of the code.
2. Verify that the model corresponds to the code.
3. Develop a formal model of the policy.
4. Use model-based testing to check that the code implements the policy.
5. Formally verify that the model implements the policy.

Our formal verification ensures that the TSE security policy maps directly to the model, and the model to the implementation. To map the policy to the model, we use the Isabelle Higher Order Logic (HOL) theorem prover [3]. The theorems we prove in this logic are the following:

- None of the error states are reachable.
- The noninterference property holds.

The noninterference property states that all system actions by high security-level components are invisible to low security-level components; that is, the final state of the low-level component is the same as it would be if no actions had occurred at the high-security level.

To map the model to the implementation, a code-to-spec review team of at least two people performs a line-by-line inspection of the HOL code and the C implementation.

The example in Table 1 – a single step of the BAC – shows how closely the model matches the implementation. Our model-based testing approach uses the QuickCheck tool [4]. Based on a formal statement of the security policy,

QuickCheck generates test cases that check whether or not the implementation violates that policy. The policies we have verified using this method are the following:

- **Read-across:** Reads fetch the data written at that same level.
- **Read-down:**
  - Valid reads succeed.
  - Invalid reads (that is, read-up) fail.
  - Read-downs do not affect the lower level being read (noninterference).

Other Key Components

MILS Separation Kernel

The BAC, when hosted by the MILS separation kernel [5, 6], is an instantiation of the *reference monitor* concept [7]. Unlike a traditional operating system that provides many services and abstractions, a separation kernel provides only data isolation among separate partitions and controlled communication between *partitions*. Porting an application to MILS also requires choosing a runtime or operating system to run within each partition that provides the higher-level system services the application requires, or porting one of your own choosing.

It is not enough simply to port a single-level application to a MILS separation kernel, however. The system needs to be thoughtfully decomposed and mapped to MILS partitions. Further, some key components (such as the file system) may need to be radically restructured to function in a multilevel environment.

While the TSE project aims to be portable across separation kernels, the initial target is Green Hills Software's INTEGRITY Server. This platform allows us to deploy software components from different security levels on the same hardware, thus reducing space, weight, and power requirements while retaining isolation properties equal to those provided by networks on physically separate hardware.

The WebDAV Server

The single-level components of the TSE are the WebDAV server, the file system, the network stack, and the secure sockets layer/transport-layer security (SSL/TLS). To provide the security aspects of WebDAV with high assurance, we implemented the WebDAV server using Haskell, a type-safe functional language [8]. We ported the Haskell runtime system to INTEGRITY server. The Haskell runtime system encapsulates services such as networking, threading, and memory management.

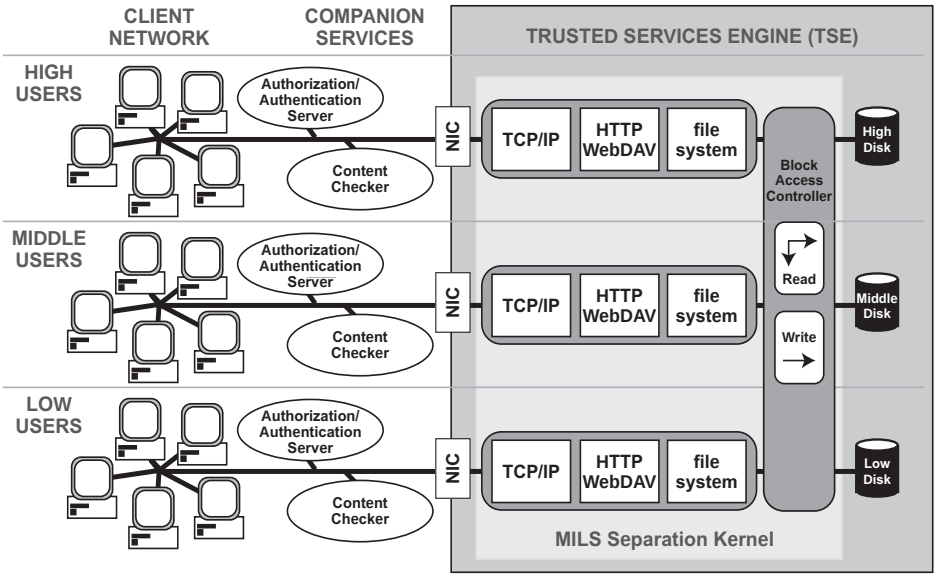


Figure 1: *Trusted Services Engine (TSE) Architecture*

The Wait-Free File System

As Figure 1 shows, the TSE file system is a single-level component. We were surprised to find that no existing single-level file system met our requirements. The problem is caused by read-down – a user on a high-level network can read files from a lower level while a user on the low-level network changes those files. Ordinarily, locks could be used to solve this problem, but cross-domain locks violate non-interference and are unacceptable in this case. How can the TSE present consistent data without introducing a proscribed communication channel, overt or covert?

Designers of algorithms for shared-memory multiprocessors face a similar problem that they solve using a method called *wait-free synchronization* [9]. Wait-free synchronization guarantees that interactions with concurrent objects take a finite number of steps instead of using *critical sections*, which block competing processes for an indeterminate time. The *wait-free file system* adapts this idea for its own synchronization method. This preserves the isolation property by the following:

- Writers are oblivious to readers.
- Readers can proceed independently of writers.

Outside Services

To minimize the trusted base and avoid duplication of function, the TSE will use, or uses outside services wherever possible. Key services are authentication and integrity-checking; so far we have evaluated Navy enterprise single sign-on for authentication and one-way file transfer for integrity-checking, but final decisions will be driven by the demands of specific installations at customer sites.

Though it is conservative and efficient to draw on outside services, it also means that we must build a chain of trust from our base to the outside service. We use several methods to help us do so:

- Outside services are all single-level, which minimizes their trustworthiness requirements.
- We choose services specified and trusted by our customers that have been vetted in similar deployment scenarios.
- The TSE and companion services use the standard cryptographic protocols SSL/TLS and digital certificates to manage communication between them.

The sum of the TSE and a specific set of external services is submitted for

Table 1: *A Single Step of the Block Access Controller*

HOL Model	C Code

bacStep :: "config => (unit, store) m"
"bacStep conf ==
let n = numLevels conf
in processQueuedLevels
(requestsPerLevel conf) n
>> queueLevels conf n"
	void bacStep (config conf) {
	nat n = conf->numLevels;
	processQueuedLevels
	(conf->requestsPerLevel, n);
	queueLevels(conf, n);
	}

the certification prerequisite to multilevel deployment.

Building Complex Multilevel Services on the TSE

The TSE can be used as a building block for more complex cross-domain services, as demonstrated by another current Galois project, the Multilevel Document Collaboration Server (DocServer). Its architecture reuses the decomposition structure of the TSE to provide multilevel secure document-based collaboration.

The DocServer allows a user at a high network level to make private modifications to an XML-based document stored at a lower level. The DocServer supports ongoing modifications at multiple network levels; modifications from the high network are visible only to users on the high network, while modifications from the low network are visible to users at that level and above.

The DocServer also supports publishing regraded documents from high network levels to low, using XML filtering and integration with an outside regrading system such as Radiant Mercury or ISSE Guard. These systems enable transfer of documents from high security to low security by enabling a human reviewer to reliably review all of a document's contents (including possibly hidden content), and, upon successful review, write it to the low network.

In the case of the DocServer, a high-level user marks up the document according to a new set of security levels, and submits it for regrading. The DocServer filters the document and sends the filtered version to the regrading system. After human review, the filtered version of the document is written to the DocServer's low-level file system.

Figure 2 shows the *publish*, *edit*, *merge* workflow of the DocServer. At left, a user on the Secret network publishes the document to the Unclassified network. The DocServer filters the Secret content and submits the resulting unclassified document to the regrader. After regrading, users on both network levels make modifications to the document. Modifications made at Secret are not visible below, but Unclassified modifications are visible to users at Secret using the DocServer's *merge* each time the document is read.

The DocServer is a Phase 1 Small Business Innovative Research project funded by SPAWAR.

Conclusion

The DocServer uses the TSE for file storage and its sole cross-domain component. Reusing the only high-assurance component gains us a great deal – the DocServer should be certifiable to the same level as the TSE with little additional work.

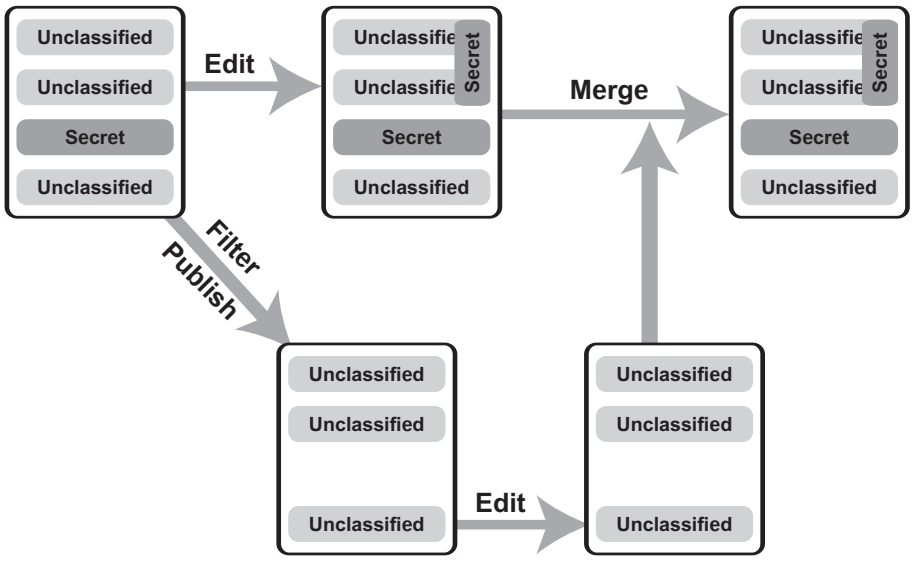
The DocServer's use of the TSE to achieve high assurance, cross-domain function mirrors the TSE's internal use of the BAC. By building the DocServer from this core component, we once again take advantage of the BAC, effectively extending its security policy through to increasingly complex systems.

The TSE's component architecture demonstrates a powerful technique for extending the security properties of a formally analyzed core component to a wide scope. In a similar manner, the DocServer uses MILS to extend the security properties of the TSE outward to provide complex multilevel functionality.

TSE Status

Development of Vers. 1.0 of the TSE will be complete in summer 2006, and will be followed by certification at a customer site. We expect to begin Common Criteria evaluation at evaluation Level 6+ the following year. Phase 1 of the DocServer is near completion. We hope to begin Phase II in spring 2006, and commercial transition sometime in 2007.◆

Figure 2: *DocServer Merge Operations*



Acknowledgements

The authors would like to acknowledge contributions from the following people: David Burke with the evaluation and certification sections; John Matthews and Paul Graunke with the verification and validation sections; and Lauren Ruth Wiener with the clarity of thought and exposition.

References

1. Common Criteria <www.commoncriteriaportal.org>.
2. D. E. Bell and L. J. LaPadula. Secure Computer Systems: Mathematical Foundations and Model. The Mitre Corporation, 1976. <http://csrc.nist.gov/publications/history/bell76.pdf>
3. Isabelle. "A Proof Assistant for Higher-Order Logic." University of Cambridge Computer Laboratory <www.cl.cam.ac.uk/Research/HVG/Isabelle>.
4. QuickCheck Automatic Specification-Based Testing <www.cs.chalmers.se/~rjmh/QuickCheck>.
5. National Information Assurance Partnership. U.S. Government Protection Profile for Separation Kernels in Environments Requiring High Robustness. Vers. 0.621. Ft. Meade, MD: NIAP, July 2004 <http://niap.nist.gov/pp/draft_pps/pp_draft_skpp_hr_v0.621.html>.
6. Vanfleet, Mark W., et al. "MILS: Architecture for High-Assurance Embedded Computing." CrossTalk Aug. 2005 <www.stsc.hill.af.mil/crosstalk/2005/08/0508vanfleet_etal.html>.
7. Anderson, James P. "Computer Security Technology Planning Study." Fort Washington, PA: James Anderson & Co, Oct. 1972 <http://csrc.nist.gov/publications/history/ande72.pdf>.
8. Haskell. Haskell: A General-Purpose Purely Functional Language <www.haskell.org>.
9. Herlihy, Maurice. "Wait-Free Synchronization." ACM Transactions on Programming Languages and Systems (TOPLAS) 13.1: 124-149. New York: ACM Press, Jan. 1991 <http://portal.acm.org/citation.cfm?id=102808>.

About the Authors

Dylan McNamee, Ph.D., is the technical lead for cross domain projects at Galois Connections. He received his doctorate in computer science from the University of Washington.

Galois Connections
12725 SW Millikan WY
STE 290
Beaverton, OR 97005
Phone: (503) 626-6616 x137
E-mail: dylan@galois.com

CDR Scott Heller is currently the Cross Domain Solutions lead at PMW 160 within the Program Executive Office Command, Control, Communications, Computers, and Intelligence, and Space in San Diego, Calif. He has a master's degree in computer science with an emphasis in Multilevel Security from the Naval Post-Graduate School in Monterey, Calif.

Program Executive Office
C41 and Space
626 Orange AVE #303
Coronado, CA 92118
Phone: (619) 929-1451
E-mail: scott.heller@navy.mil

Dave Huff serves as the director, Exploratory Projects Division at the Fleet Numerical Meteorology and Oceanography Center. His team is focused on information assurance and Web-based techniques for establishing identity, authorization, and cross-domain information exchange.

Fleet Numerical Meteorology
and Oceanographic Center
7 Grace Hopper AVE
Monterey, CA 93943
Phone: (831) 656-4569
E-mail: dave.huff@
metnet.navy.mil

Web Sites

Enterprise Software Initiative

www.esi.mil

The Enterprise Software Initiative (ESI) is a joint Department of Defense (DoD) project to develop and implement a DoD enterprise process. The objectives are to save money and improve information sharing. The initial focus will be on commercial off-the-shelf (COTS) products. The main problem identified with procuring software for DoD is that the software (including price, acquisition cost, distribution, training, maintenance, and support) costs too much. Enterprise Software is DoD common-use, standards-compliant software. The DoD ESI Steering Group, under the DoD Chief Information Officers (CIO) Council, will develop and implement a DoD Enterprise Process to identify, acquire, distribute, and manage Enterprise Software. Comprised of agencies such as the Office of the Secretary of Defense – ASD(NII)/DoD CIO, the Department of the Navy, the Department of the Air Force, the Department of the Army, the Missile Defense Agency, the Defense Finance and Accounting Service, the Defense Information Systems Agency, the Defense Logistics Agency, and the National Geospatial-Intelligence Agency, ESI follows 14 principles to ensure cost effective software procurement and provides 23 Best Practices to all Enterprise Software Agreements with the DoD and the corporate world.

Defense Acquisition University

www.dau.mil

The Defense Acquisition University (DAU) touches all areas of Acquisition, Technology, and Logistics workforce throughout all professional career stages. The DAU offers a range of basic, intermediate, and advanced certification training, assignment-specific training, performance support, job-relevant applied research, and continuous learning opportunities.

By typing <https://acc.dau.mil/simplify/ev_en.php?ID=94877_201&ID2=DO_TOPIC> into your Web browser, the Quadrennial Defense Review (QDR) 2006 Report overview page opens. The copy of the Department of Defense (DoD) QDR Report addresses key logistic and sustainment points and can be accessed at the bottom of the page by clicking <qdr2006.pdf>. The review points out successes of U.S. Transportation Command to improve the department's standard processes for providing materiel and logistics to meet the immediate needs of forces in the field. Also, the review identifies opportunities for continued transformation of acquisition and logistics processes. The QDR outlines the department's implementation of a number of specific initiatives aimed at meeting supply chain objectives.

Office of Force Transformation

www.oft.osd.mil

The Office of Force Transformation (OFT) is solely dedicated to transformation, linking creativity to implementation. OFT works at the intersection of unarticulated needs and non-consensual change, identifying and managing disruptive innovation. OFT works outside the normal course of business activities with an entrepreneurial mindset. The OFT has outlined its Top Five Goals of the Director, Force Transformation: 1) Make force transformation a pivotal element of national defense strategy and Department of Defense corporate strategy effectively supporting the four strategic pillars of national military strategy; 2) Change the force and its culture from the bottom up through the use of experimentation, transformational articles (operational prototyping) and the creation and sharing of new knowledge and experiences; 3) Implement Network Centric Warfare as the theory of war for the information age and the organizing principle for national military planning and joint concepts, capabilities, and systems; 4) Get the decision rules and metrics right and cause them to be applied enterprise wide; and 5) Discover, create, or cause to be created new military capabilities to broaden the capabilities base and mitigate risk.

Practical Performance-Based Earned Value

Paul J. Solomon
PMP

Tuesday, 2 May 2006
Track 5: 2:25 – 3:10 p.m.
Room 251 D-F

Performance-Based Earned Value's® (PBEV℠) foundation, characteristics, and guidelines were described in previous CrossTalk *articles [1] and [2]. This update includes current Department of Defense guidance on systems engineering and practical examples of implementing two of PBEV's four principles. It provides examples of basing earned value on measures of technical progress, on the progress of requirements management activities, and on the entry and exit criteria for technical reviews. This article also includes guidance for using PBEV to monitor a project.*

Performance-Based Earned Value® (PBEV℠) is a set of principles and guidelines that specify effective measures of technical performance for use with earned value management (EVM). Its guidelines are based on standards and models for systems engineering, software engineering, and project management. PBEV also supports Department of Defense (DoD) policy and guides. PBEV ensures that the product requirements baseline, or technical baseline, is incorporated into the performance measurement baseline (PMB). PBEV is an enhancement to the EVM Systems (EVMS) standard [3].

DoD Guides

DoD acquisition policy states that programs implement systems engineering plans (SEP) that include the success criteria for technical reviews [4]. DoD guides that implement the policy include the Defense Acquisition Guidebook (DAG), the Systems Engineering Plan Preparation Guide (SEPPG), the Work Breakdown Structure Handbook (MIL-HDBK-881A [WBS]), and the Integrated Master Plan and Integrated Master Schedule Preparation and Use Guide. Table 1 shows pertinent components of the guides.

The DoD guides refer to EVMS. However, EVMS has significant limitations with regard to the standards and models for systems engineering, software engineering, and project management [2]. Unless these limitations are addressed, there is no assurance that the PMB will include the activities and measures that lead to success. PBEV overcomes these limitations.

For example, the EVMS guidelines specify that earned value (EV) be based on work performed, but only indirectly link EV to meeting the product requirements or the expected quality. In comparison, PBEV bases EV on progress toward meeting the allocated product requirements. PBEV's EV is based on the sum of two measures:

- Progress toward completing the set of enabling work products.
- Progress toward meeting the product requirements.

Table 1: *Department of Defense Systems Engineering Policy and Guides*

DoD Systems Engineering Guides	DAG	SEP	WBS	IMP/ IMS
Develop Systems Engineering Plan (SEP).	4.2.3.2	1.0		
Event-driven timing of technical reviews.	4.5.1	3.4.4	3.2.3.1	2.3, 3.3.2
Success criteria of technical reviews.	4.5.1	3.4.4	3.2.3.1	3.3.2
Assess technical maturity in technical reviews.	4.5.1	3.4.4	3.2.3.1	
Integrate SEP with Integrated Master Plan (IMP).	4.5.1	3.4.5		1.2, 2.3
Integrate SEP with Integrated Master Schedule (IMS).	4.5.1	3.4.5		1.2, 2.3
Integrate SEP with Technical Performance Measurement (TPM).	4.5.1	3.4.4		1.2, 2.3
Integrate SEP with Earned Value Management.	4.5.1	3.4.5		1.2, 2.3
Integrate Work Breakdown Structure (WBS) with requirements specification, statement of work, IMP, IMS, and Earned Value Management System.			2.2.3, 3.2.3.3	3.4.3
Use TPMs to compare actual versus planned technical development and design maturity.	4.5.5	3.4.4		3.3.2
Use TPMs to report degree to which system requirements are met in terms of performance, cost, and schedule.	4.5.5	3.4.4		
Use standards and models to apply systems engineering.	4.2.2 4.2.2.1			
Institute requirements management and traceability.	4.2.3.4	3.4.4		

PBEV Principles and Guidelines

PBEV's foundation, characteristics, principles, and guidelines were previously discussed [2]. Some guidelines that are referenced in this article are included in Table 2.

PBEV Process Flow

A comparison of the PBEV process flow with the traditional EVMS process flow is shown in Figure 1. The PBEV processes and guidelines that supplement EVMS are highlighted. PBEV includes three processes that supplement EVMS that address the product requirements:

- Define the product (also called the technical baseline).
- Integrate product requirements and quality with the plan.
- Measure progress toward meeting product requirements and quality.

A fourth PBEV process addresses risk management:

- Integrate risk management with the plan.

® Performance-Based Earned Value is registered with the U.S. Patent and Trademark Office by Paul Solomon.
℠ PBEV is a service mark of Paul Solomon.

Progress Toward Meeting Requirements

Advice and examples follow for practical implementation of the PBEV guidelines that address the product requirements. The program manager (PM) should select base measures for EV that indicate progress toward development, maturity, implementation, and testing of the product requirements.

Project management processes require progress reporting at periodic intervals, normally monthly. However, progress toward meeting product requirements is not always measurable on a periodic basis. For example, a hardware or software component may require the completion and assembly of many enabling work products such as drawings or coded software modules, before the integrated set of work products may be measured against product quality objectives. Consequently, interim progress measurement is normally against the scheduled completion of enabling work products.

The first two examples apply to PBEV guidelines that address the product requirements (Guidelines 1.1, 2.2, 2.5, 2.6, and 2.7).

Example 1: EV Based on Completing Drawings and Meeting Requirements

Example 1 shows how to base EV on both progress toward completing the set of enabling work products and progress toward meeting the product requirements.

The output of a work package is the design of a component of a subsystem, a set of wire harnesses. There are two requirements that are allocated to the wire harnesses: maximum weight and maximum diameter. The requirements follow:

- Maximum weight: 200 pounds.
- Maximum diameter: 1 inch.

The progress and EV of the work package is measured by both the completion of the enabling work products (drawings) and by meeting the requirements. The schedule for completing the drawings and for meeting the requirements is shown in Table 3 (see page 22).

The budget is allocated as follows: The work package for a component has a budget at completion of 2,000 hours. Each drawing has a budget value of 40 hours.

EV is dependent on the engineering analyses that are performed to determine that the design meets the requirements. EV, also called Budgeted Cost of Work Performed (BCWP), is decreased (negative EV) if a requirement was not met on schedule. EV is restored when the requirement is finally met. The total possible negative EV is 300 hours, as follows:

- Component weight requirement not met: -100.
- Diameter requirement not met: -200.

The schedule status at April month end follows:

- Cumulative drawings completed: 41.
- Diameter requirement met.
- Component weight requirement not met.

Table 4 shows the time-phased Budgeted Cost for Work Scheduled (BCWS), how EV increases for completing the drawings and is reduced if the design fails to meet requirements.

The unfavorable schedule variance analysis should state that the drawings are ahead of schedule (+40) but the design has not met the planned requirements (-100). There will be an unfavorable impact to both the cost and schedule objectives as the drawings are reworked until the design meets the requirements.

A discussion and examples of basing EV on meeting software requirements, including a technique for quantifying deferred functionality, are provided in [1].

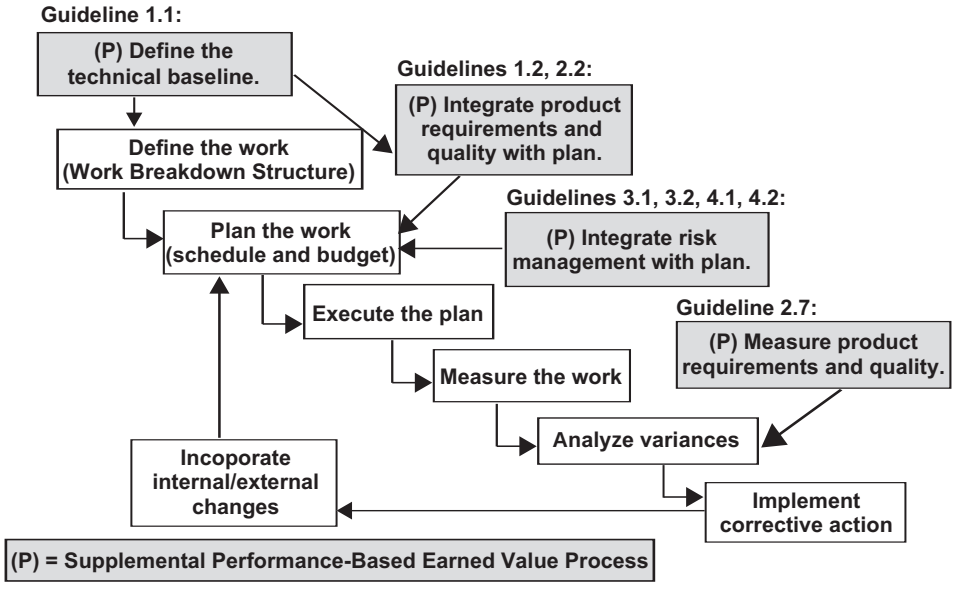
Referenced Performance-Based Earned Value Guidelines	
1.1	Establish product requirements and allocate these to product components.
1.2	Maintain bidirectional traceability of product and product component requirements among the project plans, work packages, planning packages, and work products.
2.2	Specify work products and performance-based measures of progress for meeting product requirements as base measurements of earned value. Examples are: • Results of trade-off analysis. • Allocated requirements developed, implemented into design, or tested successfully. • Achieving planned technical performance measures. • Meeting entry and success criteria for technical reviews. • Other quality objectives achieved.
2.4	Identify event-based, success criteria for technical reviews that include development maturity to date and the product's ability to meet product requirements.
2.5	Establish time-phased, planned values for measures of progress towards meeting product requirements, dates or frequency for checking progress, and dates when full conformance will be met.
2.6	Allocate budget in discrete work packages to measures of progress towards meeting product requirements.
2.7	Compare the amount of planned budget and the amount of budget earned for achieving progress towards meeting product requirements.

Table 2: *Referenced Performance-Based Earned Value Guidelines*

Technical Performance Measurement

Technical Performance Measurements (TPMs) are defined and evaluated to assess how well a system is achieving its performance requirements. TPM uses actual or predicted values from engineering measurements, tests, experiments, or prototypes. In Example 1, TPMs are used

Figure 1: *Earned Value Management Systems and Performance-Based Earned Value Process Flows*



Schedule Plan	Jan.	Feb.	Mar.	Apr.	May	Total
Drawings	8	10	12	10	10	50
Requirements Met:						
Weight					1	1
Diameter				1		1

Table 3: *Schedule for Drawings and Requirements*

to determine if the weight and diameter requirements will be met.

Often, during the early stages of drawing development, it may be too early to measure TPM progress. For tasks that are scheduled to complete before the first TPM milestone, EV would be based only on completing drawings per the organization's process quality procedures and standards. Eventually, enough drawings will have been completed to enable the measurement of TPM achievement. If a percentage of the work package budget had been allocated to completing the drawings and another percentage to achieving planned TPM values, then the work package would be held to less than 100 percent complete until the TPM planned values are achieved.

If a TPM planned value is not achieved when scheduled, take negative EV for not meeting that requirement, as was shown in Example 1.

The achievement of significant performance requirements may not be measurable at the component level. If the design of a component is at the work-package level, completion of the design may depend on achieving planned TPMs values or other quality objectives that are only measurable at a higher level of the system architecture or WBS. A technique for constraining EV for a component level work package is to earn part of the work-package budget when the performance objective is met at the higher level of the WBS.

Example 2 is typical during development of a project. A TPM objective is established at the subsystem level. Many, if not all, of the components of the subsystem contribute to technical performance. For a weight TPM, all components play a part. For other TPMs, such as response time, a subset of the components, including both hardware and software components, contributes to the subsystem objective. In Example 2, EV at the component level is based on both the weight of the component (200 pounds) and the weight of the subsystem to which it belongs.

Table 4: *Net Budgeted Cost for Work Performed Based on Component Requirements*

Design (drawings)	Jan.	Feb.	Mar.	Apr.	May	Total
Planned drawings	8	10	12	10	10	50
Budgeted Cost for Work Schedule (BCWS) – current	320	400	480	400	400	2000
BCWS – cumulative	320	720	1200	1600	2000	2000
Actual drawings completed	9	10	10	12		
Budgeted Cost for Work Performed (BCWP) (drawings) – current	360	400	400	480		
BCWP (drawings) – cumulative	360	760	1160	1640		
Negative BCWP (requirements) – cumulative				100		
Net BCWP (drawings and requirements)				1540		
Schedule variance	40	40	-40	-60		

Table 5: *Net Budgeted Cost for Work Performed Based on Component and Subsystem Technical Performance Measurements*

Design (drawings)	Jan.	Feb.	Mar.	Apr.	May	Total
Planned drawings	8	10	12	10	10	50
Budgeted Cost for Work Schedule (BCWS) – current	320	400	480	400	400	2000
BCWS – cumulative	320	720	1200	1600	2000	2000
Actual drawings completed	9	10	10	12		
Budgeted Cost for Work Performed (BCWP) (drawings) – current	360	400	400	480		
BCWP (drawings) – cumulative	360	760	1160	1640		
Negative BCWP (component weight) – cumulative				-100		
Negative BCWP (subsystem weight) – cumulative				-200		
Net BCWP (drawings and requirements with technical performance measures)				1340		
Schedule Variance	40	40	-40	-260		

Example 2: EV When TPM Is At a Higher WBS Level

The assumptions of this example follow:

- The component in Example 1 is one of four components that form a subsystem.
- The subsystem's TPM objective is 4,000 pounds.
- The SEP states that some components may be overweight at completion if there are offsets in other components as long as the total subsystem weight does not exceed 4,000 pounds.

The EV solution for the component that was first shown in Example 1 has changed. In this example, the total possible negative EV is 500 hours, as follows:

- Component weight TPM planned value not met: -100.
- Subsystem weight TPM planned value not met: -200.
- Diameter requirement not met: -200.

In this example, the EV of the work package for a component is dependent on both the measured weight of the component and the weight of the other components within the same subsystem. If both the component and the subsystem weight planned values were not achieved at the April milestone, the net BCWP would be 1,340 hours, as shown in Table 5, Net BCWP Based on Component and Subsystem TPMs. This technique may also incorporate higher levels of the WBS.

Example 3: Progress of Requirements Traceability and Verification

Guideline 1.2 addresses requirements traceability. This guideline supports the SEPPG guidance for the technical management and control section of the SEP. This section of the SEP describes the approach for controlling the overall technical effort of the program, including the technical baseline control and requirements management, traceability, and requirements verification.

Example 3 demonstrates a method for measuring progress of the systems engineering effort to perform requirements management, traceability, and verification. Typical activities include: define the requirement, validate the requirement, determine the verification method, allocate the requirement, document the

verification procedure, and verify that the requirement has been met. The requirements traceability matrix (RTM) should be used to record the status of each requirement as it progresses through this cycle. A time-phased schedule for the planned completion of these activities is the basis for the PMB. A measure of the status of the system or subsystem requirements in the RTM should be a base measure of EV.

In Example 3, a system includes five components, 16 total requirements, and six systems engineering activities. The budget allocation is shown in Table 6.

An example of the schedule and the BCWS for the systems engineering effort for one of the components, the enclosure, is shown in Table 7. The time-phased BCWS is determined by allocating the budget for each activity to the month in which it is scheduled.

Software Engineering Budget	Number of Requirements	Software Engineering Budget	Define	Validate	Verify Methods	Allocate	Verify Document	Verify
Budget Percent			15%	15%	15%	20%	15%	20%
Component								
Enclosure	3	240	36	36	36	48	36	48
Transmitter	1	80	12	12	12	16	12	16
Battery	2	160	24	24	24	32	24	32
Control	1	80	12	12	12	16	12	16
Software	9	720	108	108	108	144	108	144
Total	16	1280	192	192	192	256	192	256

Table 6: *Systems Engineering Budget Allocation*

Using PBEV to Monitor a Project

A customer may use PBEV to validate the planning baseline and to monitor the supplier's progress. The customer should utilize the Integrated Baseline Review (IBR) to verify that the SEP includes all required plans, planned values, and process descriptions. The IBR should also be used to verify that the plans, entry criteria, and exit criteria in the SEP are integrated with the master schedule and the work packages. For example, the master schedule should include the criteria for completing technical reviews and milestones for measuring technical performance as well as the TPM planned value to be achieved at that milestone.

Example 4: Exit Criteria

The entrance and exit criteria for event-driven technical reviews should be defined in the SEP. The exit criteria should also be the completion criteria for work packages that map to the reviews. An example of the exit criteria for a system-level detailed (critical) design review, from the systems engineering standard, Institute of Electrical and Electronics Engineers (IEEE) 1220-1998 [5], follows:

- Detailed design satisfies system baseline.
- Design solution meets the following:
  - Allocated functional and performance requirements.
  - Interface requirements.
  - Workload limitations.
  - Constraints.
- Design verification complete for the following:
  - Each requirement constraint is traceable to the physical architecture.
  - Design element solutions satisfy the validated requirements baseline.

PBEV guidelines 2.2 and 2.4 address technical reviews. The customer should apply these guidelines when reviewing the SEP with the supplier. Use the IBR to reach agreement on the entry and exit criteria for all major technical reviews with regard to the technical baselines. The technical baselines are important work products that should be included in the IMS and work packages. The technical reviews described in the DAG with their respective baselines and their IEEE 1220-1998 equivalents are shown in Table 8, DoD Technical Reviews and Baselines.

Following the IBR, the customer is advised to conduct periodic reviews to ensure suppliers are following their plans, procedures, and standards (including those for systems engineering and EVM). The customer should also perform independent assessment of the supplier's progress and verify that the correct base measures are specified and used for EV. The PM should address technical maturity, including TPM achievement and reporting, during technical assessment reviews. Finally, the PM should verify that the supplier has met the exit criteria of event-driven technical reviews.

On a recurring basis, the customer should monitor supplier reports. Review the supplier's EV reports, master schedule, and technical reports to determine if they are consistent; and evaluate supplier metrics (product, schedule, EV) by understanding and questioning the information, including variance analysis. If

Table 7: *Systems Engineering Schedule and Budgeted Cost for Work Scheduled*

	Jan.	Feb.	Mar.	Apr.	May	June	July	Total
Enclosure Schedule								
Defined	3							
Validated		2	1					
Verified Method			1	2				
Allocated					3			
Traced to Verification						3		
Verified							3	
Budgeted Cost for Work Scheduled								
Defined	36							36
Validated		24	12					36
Verified Method			12	24				36
Allocated					48			48
Traced to Verification						36		36
Verified							48	48
Total	36	24	24	24	48	36	48	240

Technical Review	Technical Baseline	DAG	IEEE 1220-1998
System Functional Review	System Functional Baseline	4.3.3.4.3	Validated Requirements Baseline
Preliminary Design Review	System Allocated Baseline	4.3.3.4.4	Verified Physical Architecture
Critical Design Review	System Product Baseline	4.3.3.4.5	Verified Physical Architecture
Production Readiness Review	System Product Baseline	4.3.3.9.3	Verified Physical Architecture

Table 8: *Department of Defense Technical Reviews and Baselines*

the information appears inconsistent or if the variance analysis and corrective action plans are insufficient, conduct reviews to obtain insight into metrics and to better understand the causes and impacts of the variances.

Conclusion

PBEV supplements traditional EVMS with the best practices of systems engineering, software engineering, and project management standards and models. Its principles and guidelines enable true integration of project cost, schedule, and technical performance.◆

References

1. Solomon, Paul J. "Practical Software Measurement, Performance-Based Earned Value." CROSSTALK Sept. 2001: 25-29 <www.stsc.hill.af.mil/crosstalk/2001/09/solomon.html>.
2. Solomon, Paul J. "Performance-Based Earned Value." CROSSTALK Aug. 2005: 25-26 <www.stsc.hill.af.mil/crosstalk/2005/08/0508solomon.html>.
3. Government Electronics and Information Technology Association. "ANSI Earned Value Management System (EVMD) Standard." ANSI/EIA-748-A-1998 R2002. Arlington, VA <http://electronics.ihs.com/abstracts/geia-standards.jsp>.
4. Wynne, Michael. Policy for Systems Engineering in DoD. Memorandum, 20 Feb. 2004. Acting Undersecretary of Defense, Acquisition, Technology and Logistics.
5. Institute of Electrical and Electronics Engineers. "IEEE Std. 1220-1998 Standard for Application and Management of the Systems Engineering Process." IEEE, Dec. 1998.

Paul J. Solomon will also be presenting a tutorial on integrating systems engineering with earned value management at the SSTC on Monday, 1 May from 8:00 to 11:15 a.m. in room 251 D-F.

About the Author

Paul J. Solomon monitors Earned Value Management Systems (EVMS) for Northrop Grumman Corporation Integrated Systems. He has supported the B-2 Stealth Bomber, Global Hawk, and F-35 Joint Strike Fighter programs. He is an author of the EVMS standard, and received the Department of Defense's David Packard Excellence in Acquisition Award. While a Visiting Scientist at the Software Engineering Institute, he authored "Using CMMI to Improve EVM." His book, "Performance-Based Earned Value," co-authored with Ralph Young, will be published by the Institute of Electrical and Electronics Engineers Computer Society. Solomon is a Project Management Professional. He has a Bachelor of Arts and Master of Business Administration from Dartmouth College.

Northrop Grumman
Integrated Systems
One Hornet WY
TD21/2C
El Segundo, CA 90245
Phone: (310) 335-3308
E-mail: solomonpbev@msn.com

Lessons Learned Using Agile Methods on Large Defense Contracts

Paul E. McMahon
PEM Systems

Monday, 1 May 2006
Track 2: 3:55 – 4:40 p.m.
Ballroom B

While the agile movement began on small commercial projects, many contractors are employing these methods today (to varying degrees) on large defense contracts. In the process, new challenges are being faced that are not addressed by current published agile literature. Examples of questions being asked include: How do we treat firm requirements? How do we report earned value? How are systems engineering, configuration management, and our test group affected? How should we handle traditional customer deliverables? What can we do about personnel who are not motivated to work on self-directed teams? This article employs scenarios based on actual project situations occurring in 2005 to share the latest lessons learned on what is working and what isn't working when applying agile software development on large government defense projects.

In a May 2005 CROSSTALK article [1], I discussed six agile software development myths and four recommended extensions to apply agile on large distributed projects. Over the past year, I have had the opportunity to work with multiple clients applying agile – or a modified form of agile – on large U.S. defense contracts. In this article, I share what was learned through nine scenarios developed from actual project experiences, along with 22 related lessons learned.

As background for those unfamiliar with agile methods, and to set the context for the scenarios, the agile manifesto [2] provides the following four value statements agreed to by the founders of the most popular agile methods:

- We value individuals and interactions over processes and tools.
- We value working software over documentation.
- We value customer collaboration over contract negotiation.
- We value responding to change over following a plan.

Scenario 1

Agile Planning

An appealing characteristic of agile software development is its potential to help manage change. A client said to me, "We have good requirements for what we know today, but technology changes fast. I need my contractor to be ready to change direction." My client's contractor had won the job based on his proposed agile approach. I was asked by my client to assess that approach.

The contractor was planning incremental deliveries of the refined and allocated requirements along with functional capabilities. I became concerned with the approach based on responses I was getting to one particular question: "What if at the start of the third increment, your customer gives you new priorities and wants to change direction?" The most common response was, "There is no room in our schedule to change direction. We already have too much to do." I then asked, "What if some things were taken off your current list?" The response was not positive, so I asked my client a similar question. He replied, "I never take anything off the list."

Analysis

The first concern is the statement that there is "no room in our schedule to change direction." Adjusting the plan continually is a fundamental characteristic of agile methods. In Scenario 1, work was partitioned into scheduled blocks called increments, but there was no real plan to adjust the effort, degree of detail, or planned tasks during each increment's detailed planning based on new priorities or risks.

The second concern was the statement "I never take anything off the list." Collaboration means cooperation, but it also implies a willingness to honestly consider alternatives. The customer, although he wants his contractor to be ready for change, does not appear to be planning to collaborate.

Agile Planning Insight

There should always be things you can take *off the list*, but this does not mean customers on agile projects must live without all their requirements. I will explain this further later in the article.

Scenario 2

Agile Requirements

I was called in to help a large agile project that was in trouble. The program manager wanted his team to be agile; to help, he initiated a few rules. His first rule for his systems engineers was, "Don't write more than 100 requirements."

When I talked to a developer on the project, he said, "We wanted more details. There was too much ambiguity in the requirements." Another said, "There was a lack of flow-down of requirements from systems engineering."

When I shared the developer's comments with a systems engineer, he said, "We were told to pull back." Then he added, "I don't get it. How are you supposed to handle firm requirements on an agile project? If we don't write detailed requirements with agile, what are systems engineers expected to do?"

Lessons Learned

The first lessons learned address these statements: Don't write more than 100 requirements, and how are you supposed to handle firm requirements on an agile project?

Lesson 1: Write down all your must-do/firm requirements as soon as you know them, and do not plan to *collaborate* on them. For those who claim this recommendation is not agile, I say, this is *practical agility* and the following is why: Trying to *collaborate on truly firm* requirements will only frustrate your team and waste resources. I have witnessed this frustration on multiple occasions during this past year.

When I use the term *collaborate*, I mean an honest consideration of alternatives and a willingness to give. I am not saying do not talk to your customer about the requirements, but I am saying if there is no room to give, then do not pretend there is. Collaboration – in the agile con-

text – means more than talking – it implies taking action that leads to change.

Some have suggested that the term *negotiation* might be more appropriate in this context, but negotiation brings with it an *us* and *them* implication. Alistair Cockburn tells us that "In properly formed agile development, there is no 'us' and 'them,' there is only 'us' [2]."

However, one cautionary note: Are you sure you recognize a must-do requirement when you see one? As an example, one of my clients is modernizing a legacy system. There are lots of firm requirements. The functionality of the legacy system must be maintained, but the users do not need to achieve that functionality the same way. This has been a great point of confusion and contention on the project.

Lesson 2: We often confuse *nice-to-have* requirements with firm *must-do* requirements. Yes, this sounds like basic systems engineering, and I know some of you may be thinking this is not an agile issue. But it is, and lesson No. 3 is why.

Lesson 3: Systems engineering is still required with agile development. I find that in the name of agile, many large projects are forgetting fundamental systems engineering.

Lesson 4: We must get out of the sequential waterfall mentality – this is an outdated way of thinking and it does not work with agile methods.

Lesson 5: Agile does not require fewer written requirements. It does require collaboration to identify the needed detail to implement what the team is focusing on now in this increment. The word *flow-down* implies an ordering – something occurring before something else. Systems engineering does its job before software developers do theirs. Systems engineering does the requirements. The developers *wait* for the *handoff*. This way of thinking will not work with agile methods.

Systems engineering *pullback* is exactly the reverse of what should be happening. On large projects in particular, there are still some very important sequential activities that must happen. For example, systems engineering must do a high-level first pass of requirements and allocate them to major incremental releases *before* the developers get going. This is by no means the end of systems engineering. Today, this point is too often being missed. The critical and most intense part of systems engineering with agile is still ahead *after* the high-level requirements. This is the collaboration on the details that must happen *concurrently,* working closely with the developers in each increment.

The No. 4 and 5 lessons learned address the following statements:

- There is a lack of flow-down of requirements.
- We were told to pull back.
- What are systems engineers expected to do?

Scenario 3

Customer Collaboration and the Program Manager

A systems engineer on a large agile project told me that he had been told to keep quiet at a review concerning a specific technical topic. He said the program manager had told him, "We want to be collaborative."

> *"If you are the program manager on an agile project, expect more conflict early ... Because of this early increased conflict, it is critical to have a strong conflict management process in place and personnel trained in using that process."*

I was concerned when I heard this comment that someone had misunderstood collaboration, so I raised the issue with another team member. He explained to me that the technical topic had been thoroughly discussed and resolved at a previous meeting. The program manager apparently did not want to waste time revisiting it. This made sense to me, but do not dismiss this scenario lightly. Effective implementation of collaboration on agile projects is closely linked to requirements lists, task lists, and collaboration rules. This is explained further in the following paragraphs.

Lessons Learned

Lesson 6: The program manager should not assume the agile team knows how to collaborate. Many will need to be taught how to recognize good collaboration opportunities, and when it is time to stop collaborating. In Scenario 3, the program manager knew the technical topic had been previously discussed and resolved. He also knew that you can collaborate too much, which led to his decision.

If you are the program manager on an agile project, expect more conflict early. This is because of the shorter iterations and risk focus. Because of this early increased conflict, it is critical to have a strong conflict management process in place and personnel trained in using that process [3]. I have observed in the past year both too much and too little collaboration.

One reason people fail to collaborate is because it can be draining. Collaboration takes time and energy. This is one reason why it is important to distinguish truly *firm* requirements from *nice-to-have* requirements. This helps us pick our battles wisely.

Recognize opportunities for effective collaboration. As an example, when a developer says, "We wanted more details," as we saw in Scenario 2, this is a likely opportunity for collaboration. He is saying we need more discussion and action (e.g., updates to task lists) because the current requirements/task *list* is ambiguous, or is missing tasks.

Lesson 7: The program manager's role on an agile project is affected by how he/she interacts with the agile team, particularly with respect to requirements and task lists. Some program managers have asked, "Does agile affect my job?" If you are the program manager, I recommend that you encourage your team to resolve ambiguous requirements and add missing tasks to the appropriate *list*. This will ultimately provide more accurate visibility of the real status back to you.

Program managers should also let their team know they expect to hear about more issues early and that they will not *shoot the messenger*. This may sound trite, but the manner in which a program manager responds to issues raised early can set the tone for the entire project with respect to timely and accurate reporting up the chain. This is particularly important on agile projects due to the increased tendency to push work out as we will see later in Scenario 5.

Lesson 8: On large projects it is necessary to have multiple lists. The organization of most large agile projects includes many *hub-teams* that interact differently from teams in traditional hierarchical organizations. We refer to the teams as hub-teams rather than *sub-teams* because of the manner in which the teams interact [1, 4].

Large projects tend to have more complex products and sub-products. This implies multiple lists. The top list includes the end-customer requirements (e.g., product backlog list for full project/product). Lower lists are more solution (e.g., design) and task oriented and are used by individual hub-teams to remove ambiguity of higher-level requirements and clarify task responsibility.

Lesson 9: When you understand how the full family of lists works together on a large agile project, you will understand why there is always something you can *take off the list*. We remove ambiguity by collaborating and adding solution space items to lower lists (e.g., hub-team lists for specific increments/iterations). By solution space items, I mean tasks associated with design decisions (e.g., the look and feel of a user interface), and other real work that team members must do (e.g., preparation for a customer review and documentation). Because the solution space provides choice, it also provides opportunity to collaborate – to consider and be open to alternatives. This is a full team responsibility and must include systems and software engineering and customer representatives.

You can think of the lower lists as the result of *successful collaboration* as long as those lists represent the real work being done by the project teams. Watch for warning signs of *failed collaboration* such as work that is happening, but is not on any list and has not been agreed to.

Scenario 4

Customer Collaboration and User Conferences

One of my agile project clients has a very large customer community. To gain early feedback, user conferences were held to demonstrate incremental versions of the product. Developers were sent to the conferences to interact directly with the users.

One team member who attended a user conference commented, "We thought the direct interaction between the customers and our developers would lead to fewer requirements, but the users wanted more." Another said, "Many users wanted different things. Our developers did not know who to listen to." Another said, "Some users became upset because they did not see all their requirements in the demonstrated product."

Lessons Learned

I used to say, "You can involve the customer too early even on an agile project." But this does not communicate the situation accurately. I now say, "You can never bring the customer in too soon as long as you know who your customer is."

Ken Schwaber, co-developer of the Scrum process (a popular agile method), uses the term *product owner* rather than *customer*. The product owner is responsible for representing the stakeholders [5]. The product owner manages the team list. In Scrum, the team list is referred to as the *Product and Sprint Backlogs*. The product owner is responsible to keep the list in priority order, and provides clarifications to the team when needed.

Lesson 10: Each hub-team must have its own *single* product owner and its own *single* list for the work that is approved to be working on now. In Scenario 4, the developers did not know who their product owner was. User conferences are encouraged to allow developers to hear the needs of end users directly. However, approval for work by individual hub-teams must be coordinated through a single product owner. Similarly, once work is approved for a hub-team, its priority must be clear and in which increment it is approved to be worked on. There should be a single list for each hub-team for the current approved work. Large projects will have many hub-teams (e.g., a project with 500 people could have 50 hub-teams). This implies 50 product (or sub-product) owners (one for each team). This does not mean each product owner must be dedicated full-time to the product owner role.

"Thinking of a systems engineer as a product owner should not seem like a foreign idea. In many large organizations, systems engineering is viewed as the customer for software engineering."

I have heard some say that agile will not scale up because there are not enough *customer* personnel. The implication is that *customer* must be the *end-customer*. But often on large agile projects, the right customer is not the end-customer, but rather someone who represents the end customer.

Lesson 11: The right systems engineer may be the perfect candidate for a product owner role. This lesson addresses the question, "What are systems engineers expected to do?" Thinking of a systems engineer as a product owner should not seem like a foreign idea. In many large organizations, systems engineering is viewed as the customer for software engineering.

Schwaber, in describing the relationship of the team and the product owner, refers to "constantly collaborating, scheming together about how to get the most value for the business" [5]. This is the model of how systems engineering, software engineering, and support organizations in large companies should be operating for effective agile operations – scheming together (in a positive way) with a common goal of value for the business. Unfortunately today, many large organizations do not operate under this model, but rather with a *throw-it-over-the-wall/not-my-problem* sequential/waterfall mind-set.

Scenario 5

Risks and Priorities

On one project, a hub-team lead engineer said he learned for the first time, in a recent formal program review with the customer, that some work his hub-team was dependent upon was being shifted out to a later increment by another hub-team on the project. That other hub-team had decided they had higher priority and higher risk tasks to work on. No one from that hub-team had coordinated the change with the dependent hub-team, nor did the lead of that team realize the impact of his team's decision.

Lessons Learned

Lesson 12: Hub-teams on larger projects must not decide to move functionality out without collaborating with their product owner. In Scenario 5, the hub-team made a decision to reprioritize their work without coordinating this change with a dependent team. The product owner must approve any changes to hub-team plans.

Lesson 13: Product owners on large agile projects must meet regularly to coordinate hub-team changes with interfacing product owners. On large projects, the product owner has a larger set of responsibilities than on small agile projects. The product owner must coor-

dinate any decisions to change priorities of planned work with all dependent product owners. Individual teams may not be aware of the full project impact of a change to their plans. The coordination process described in this lesson is missing today on many large projects attempting to be agile. Refer to Figure 1 for a diagram of a large agile project team's roles, lists, and interactions.

Lesson 14: A project integration plan is a critical artifact that needs to be employed by product owners on large agile projects. I asked a developer on a large agile project where I could find the project integration plan. He replied, "We do use cases. We do not need an integration plan."

Integration occurs earlier and more frequently on an agile project. Part of the expanded responsibilities of the product owner on large agile projects includes coordination and approval of changes to the work at the hub-team level that may have an impact on interfacing teams. A project integration plan becomes more critical on agile projects due to the increased integration frequency. It is a critical artifact that should be employed by hub-team product owners when discussing potential changes to planned work.

One reason the integration plan is so important on large agile projects is because one can become lost in the details of all the individual team lists on a large project. The integration plan helps the project leaders see the big picture, which is essential when considering plan changes.

Lesson 15: Use cases are not a replacement for the integration planning. Use cases can help developers understand the project requirements. An integration plan conveys the overall project road map, including the planned sequence of activities and dependencies. One cannot replace the other.

Scenario 6

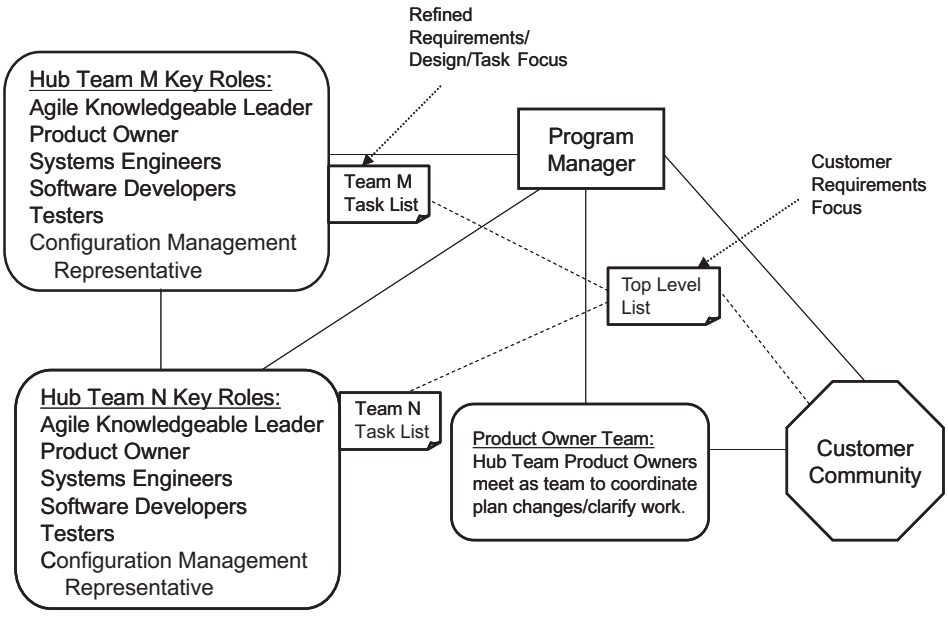
Agile Earned Value

That same hub-team that had shifted planned work out had also reported that it had completed 100 percent of its planned functionality for the same increment. When questioned about the functionality that was being moved out, the lead engineer said that his team had made the decision to move that work out based on priority and risk, so he decided not to include it in his measurement reporting for that increment.

Lessons Learned

Lesson 16: One of the greatest values of agile is early visibility to management of accurate status. This visibility is possible only if progress is reported relative to the baseline plan. In Scenario 6, the hub-team lead engineer made the decision not to include planned work in his measurements. As previously discussed, the hub-team should not make decisions on moving work without coordination with the product owner. But even if work is agreed to be moved out after the start of an increment, it is critical that the earned value report continue to be based on the original baseline plan. Key to agile is the reporting of true team velocity. A common, but costly, mistake on many large incremental projects is pushing planned work out and not raising the visibility. If you push work out, do not hide it. Raise it up through accurate earned value reporting.

Figure 1: *Large Agile Project Team Roles, Lists, and Interactions*



Scenario 7

Self-Directed Teams

I was explaining how self-directed teams operate to a group of senior leaders at one of my client's locations where they were initiating a new agile project. An experienced senior engineer interrupted with the statement, "It will never work on large projects because you will never find enough people with the necessary self-direction skills." My first reaction was that he might be right. I have since changed my view.

Lessons Learned

Lesson 17: Seed your hub-teams with agile-knowledgeable leaders. I used to buy into the idea that agile methods required special skills that many *average* developers could not master. An example is estimating the personal effort required to complete a task, and reporting actual personal progress accurately. Watching agile take hold in organizations has led me to change this belief. Now I believe most team players can pick up agile skills easily.

When a project has leaders who understand agile practices, and mentor others by example, a self-directed culture can take hold quickly. When new developers are exposed to an effective self-directed culture, they learn by watching peers and then *just do it*. I have witnessed this rapid behavior change. It is the leadership and team culture that leads to agile success, not some special set of individual skills.

Scenario 8

Agile Customer Deliverables

Scenarios 8 and 9 are admittedly exaggerations, but they are included to communicate issues commonly faced on large defense projects trying to become more agile.

When I use the term *customer deliverables,* I mean contractually required documentation, reviews, and products (e.g., code).

The program manager on an agile project asks one of his developers, "Can you show me your documentation?" The developer responds, "We're agile, so I am not focusing on my documentation." The program manager replies, "I thought you were writing agile documentation?" The developer replies, "I am, but you wouldn't

want to look at it because it is full of errors."

Lessons Learned

Lesson 18: Agile deliverables must be determined collaboratively with the customer early. Cockburn tells us that when it comes to determining what should be in a document, the answer is whatever the sponsor and the team decide [6].

Too often, I see a lack of discussion on customer deliverables early with the customer on large agile projects. So we should not be surprised when a customer becomes upset when the early deliverables do not meet expectations.

Agile customer deliverables should not be confused with low quality deliverables. As the exaggerated Scenario 8 points out, when we do not plan our deliverables through collaboration with the customer early – and allocate time and tasks based on all the work required – the deliverables will suffer and low quality should not be a surprise.

Lesson 19: The major difference between agile deliverables and traditional milestone deliverables is what takes place before the milestone delivery. When using a traditional waterfall model, it is not uncommon for customers to see deliverables for the first time at a major project milestone. With agile, the milestone should become a non-event because it is the culmination of an on going, close working relationship between customer and contractor. But do not be tempted to delete the *milestone event*. It is necessary on large agile projects to have checkpoints to ensure collaboration is really happening.

Scenario 9

Agile Test and Configuration Management

A manager on an agile project says, "With agile, we get more for less so let's plan on doing less testing." Another manager on the project replies, "Okay, and let's keep the testers and configuration management people in a separate building so they do not slow the agile team down."

Lessons Learned

Lesson 20: Agile does not always mean less. For example, do not plan on less testing. With agile, we do less of certain activities because other activities compensate. For example, we may do less formal written detailed requirements partly because the detailed test cases can compensate [7]. With agile, we test continuously to ensure previous iterations function properly along with new functionality. With agile, it is flawed thinking to believe you can do less testing.

Lesson 21: Testers must be part of the hub-teams. Agile developers must do their own low-level testing due to the tight coupling of the test-code-design cycle. Distinct testers on large projects writing higher level tests must work closely with developers to ensure complete test coverage. In our exaggerated Scenario 9, the testers were placed in a different building partly to keep from slowing the agile team down and partly to provide a level of test independence. On large agile projects, you can still have an independent group run tests, but this does not mean they should be physically separated from the team.

Lesson 22: Configuration management must be integrated into the hub-teams. Cockburn tells us that the configuration management system is steadily cited by teams as their most critical non-compiler tool [6]. This is partly because of the systems support for individual check-in, check-out, and continuous integration. On large agile projects, I have found another reason why configuration management must be integrated into each hub-team.

Schwaber uses the term *shippable* [5] in describing the quality that each iteration's product must have. With agile, we must never demonstrate to the customer a product that has not been fully tested and is *ready to ship*, even if we do not plan on deploying it today.

The reason is visibility – accurate reporting. What we demonstrate must be *done*. If it is not done, we do not report it as done, and we do not demonstrate it. This is an essential practice of agile methods.

Done means ready to ship, which means fully tested, documented, and supportable. If it is not done, do not pretend it is. Configuration management, especially on large agile projects, can provide an important checkpoint to keep the team from caving in on their definition of done when external pressures mount.

Conclusion

Many of the lessons discussed in this article are not new, and some may appear to have little – if anything – to do with agile. Examples include the following: distinguishing *must-do* require-

ments from *nice-to-have*; providing more details for ambiguous requirements; providing single focal points (product owners) for work and work change approval; coordinating schedule changes across the project; developing and using an integration plan; reporting earned value relative to your baseline plan; determining how collaborating teams resolve conflict; planning the work and scheduling the time for customer deliverables; and controlling your baseline releases through your configuration management system.

While it may appear that these are not agile issues, they are very real agile issues. This is because today – in the name of agility – we are witnessing a breakdown of fundamental systems engineering.

Agile is not a short-cut around systems engineering. It is not about systems engineers stepping back and letting developers go. It is about systems engineering stepping forward and working more effectively with all project stakeholders. It is about implementing more effective ways to manage our work *lists* and communicating the results more effectively.

Ultimately, agile is about value achieved through managed change. In particular, make small changes early and often so we do not get surprised by the big ones later.◆

References

1. McMahon, Paul. E. "Extending Agile Methods: A Distributed Project and Organizational Improvement Perspective." CrossTalk May 2005 <www.stsc.hill.af.mil/crosstalk/2005/05/0505mcmahon.html>.
2. Cockburn, Alistair. Agile Software Development. Addison-Wesley, 2002: 215-218.
3. McMahon, Paul. E. Virtual Project Management: Software Solutions for Today and the Future. St. Lucie Press, 2001: Chapter 5.
4. Highsmith, Jim. Agile Project Management. Addison-Wesley, 2004: 239-240.
5. Schwaber, Ken. Agile Project Management with Scrum. Microsoft Press, 2004.
6. Cockburn, Alistair. Crystal Clear: A Human-Powered Methodology for Small Teams. Addison-Wesley, 2004: 178, 37.
7. Ambler, Scott. Agile Modeling. John Wiley & Sons, 2002: 217.

About the Author

Paul E. McMahon, principal of PEM Systems, helps large and small organizations as they move toward increased agility. He has taught software engineering, conducted workshops on engineering process and management, published articles on agile software development, and is author of "Virtual Project Management: Software Solutions for Today and the Future." McMahon is a frequent speaker at industry conferences including the Systems and Software Technology Conference, and is a certified ScrumMaster. He has more than 25 years of engineering and management experience working for companies including Hughes and Lockheed Martin.

PEM Systems
118 Matthews ST
Binghamton, NY 13905
Phone: (607) 798-7740
E-mail: pemcmahon@acm.org

Transform This

If you are in the defense business and not heard of transformation, you are not in the defense business. Transformation is everywhere. The Army is transforming, the Navy is transforming, the Air Force has a flight plan for transformation and not to be outdone, NATO has a transformation command.

Transformation is the new clarion call to change; this year's TQM, MBO, or BPI, times ten. Do you want funding? Start transforming. Do you want to get promoted? Get involved in transformation.

This issue of CROSSTALK, coinciding with the Systems and Software Technology Conference, focuses on transforming business, security, and warfighting. We are transforming business, systems, organizations, and the force. How long will it be until the Jedi slogan transmogrifies to, "May the Force be Transformed."

So, what are we transforming into? Transformation for transformation's sake is not always a good idea. Michael Jackson transformed and look how that turned out. My fear is the siren for transformation is drowning out the end goal of the transformation. A key indicator shows up on a majority of the transformation Web sites and organization road maps, stating, "Transformation is foremost a continuing process that does not have an end point." Modifying an old proverb; with no end point any road will do.

Let us venture back to our early math classes and use transformation matrices to illustrate a few points. For those who have not been transforming matrices lately, below is a refresher on matrix multiplication.

$$\begin{bmatrix} C_{11} & C_{12} & C_{13} \\ C_{21} & C_{22} & C_{23} \\ C_{31} & C_{32} & C_{33} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} & B_{13} \\ B_{21} & B_{22} & B_{23} \\ B_{31} & B_{32} & B_{33} \end{bmatrix}$$

where,

$C_{11} = A_{11} B_{11} + A_{12} B_{21} + A_{13} B_{31}$ $\quad C_{12} = A_{11} B_{12} + A_{12} B_{22} + A_{13} B_{32}$
$C_{13} = A_{11} B_{13} + A_{12} B_{23} + A_{13} B_{33}$ $\quad C_{21} = A_{21} B_{11} + A_{22} B_{21} + A_{23} B_{31}$
$C_{22} = A_{21} B_{12} + A_{22} B_{22} + A_{23} B_{32}$ $\quad C_{23} = A_{21} B_{13} + A_{22} B_{23} + A_{23} B_{33}$
$C_{31} = A_{31} B_{11} + A_{32} B_{21} + A_{33} B_{31}$ $\quad C_{32} = A_{31} B_{12} + A_{32} B_{22} + A_{33} B_{32}$
$C_{33} = A_{31} B_{13} + A_{32} B_{23} + A_{33} B_{33}$

THE IDENTITY MATRIX: Multiplying a matrix by the identity matrix (below) yields the original matrix or no transformation.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix}$$

You go through the same steps, expend the same amount of energy, and end up where you started. When transforming, do not confuse motion with action.

THE SHEARING MATRIX: Multiplying a matrix by a shearing matrix (below) slants the original matrix parallel to the x or y-axis. A vertical slant (left) is similar to a bob.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & k \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \qquad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ k & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

A horizontal slant (right) is similar to a weave. Struggling organizations tend to bob and weave around a productive transformation with their own slant on change.

THE ROTATION MATRIX: Multiplying a matrix by the rotation matrix (below) rotates the original matrix by an angle θ counterclockwise about the origin.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

While an essential part of transformation, rotation has one danger. Used too often at the same angle, the rotation matrix spins you in a circle, like a dog chasing its tail. Do you have organizations chasing their transformation tails?

THE REFLECTION MATRIX: Multiplying a matrix by the reflection matrix (below) reflects a vector about a line (u_x, u_y) that goes through the origin.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 2u_x^2 - 1 & 2u_x u_y \\ 2u_x u_y & 2u_y^2 - 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Distorted and subdued, reflections are imperfect apes of a known solution. A good transformation should stretch, challenge and revolutionize.

AFFINE MATRICES: Adding rows and columns to a matrix allows one to mix different types of matrices.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Unfortunately, affine matrices can be used to puff up a transformation effort, making it appear more complex and effective than it truly is. Known as the peacock effect, these cosmetic add-ons drain resources with little return.

I agree with the late Vice Admiral Arthur K. Cebrowski, "The overall objective of these [transformation] changes is simply—sustained American competitive advantage in warfare." However, I suggest we tone down the platitudes to transformation itself and turn up the objectives of transformation, be it flexibility, speed, adaptability, etc.

Barney Fife, may he rest in peace, is capable of instigating transformation. People need directions. Leaders need to lead. Warriors and their supply chain need goals. Once achieved, they can set new goals but without them, I'm afraid you will get more bobbing, weaving and tail chasing than improvement.

— Gary A. Petersen
Shim Enterprise, Inc.
gary.petersen@shiminc.com

CrossTalk is co-sponsored by the following organizations:

76th SOFTWARE MAINTENANCE GROUP

309th SOFTWARE MAINTENANCE GROUP

402D SOFTWARE MAINTENANCE GROUP

Homeland Security

NAVAIR

CrossTalk / 309 SMXG/MXDB
6022 Fir AVE
BLDG 1238
Hill AFB, UT 84056-5820

PRSRT STD
U.S. POSTAGE PAID
Albuquerque, NM
Permit 737